DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Processamento Remoto em Solução para Interação com Ambientes Arquitetônicos 3D através de Tablets"

ALUNO: Guilherme Picanço Rabello
ORIENTADOR: Prof. Dr. Cesar Augusto Camillo Teixeira

São Carlos
Mar/2016

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROCESSAMENTO REMOTO EM SOLUÇÃO PARA INTERAÇÃO COM AMBIENTES ARQUITETÔNICOS 3D ATRAVÉS DE TABLETS

GUILHERME PICANÇO RABELLO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes

Orientador: Prof. Dr. Cesar Augusto Camillo Teixeira

São Carlos – SP
Março/2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Rabello, Guilherme Picanço
R114p Processamento remoto em solução para interação com ambientes arquitetônicos 3D através de tablets / Guilherme Picanço Rabello. -- São Carlos : UFSCar, 2016.
112 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2016.

1. Computação gráfica. 2. Mídia de alto desempenho. 3. Interface remota. 4. Renderização remota. 5. Plataforma móvel. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Guilherme Picanço Rabello, realizada em 07/03/2016:

Prof. Dr. Cesar Augusto Camillo Teixeira
UFSCar

Prof. Dr. Luis Carlos Trevelin
UFSCar

Prof. Dr. André Santanchè
UNICAMP

Dedico este trabalho às pessoas mais presentes na minha vida:

Minha mãe, Elisete, pelo exemplo de persistência.

Meu pai, Gisberto, pelo exemplo de descontração

Minha irmã, Jaqueline, pelo exemplo de companheirismo

AGRADECIMENTOS

Meus sinceros agradecimentos

A meus pais Elisete e Gisberto, por sempre me apoiarem e acreditarem em minhas escolhas em todos os momentos importantes de minha vida.

A minha irmã, Jaqueline, pela parceria e ajuda em muitos momentos difíceis pelos quais passei durante a realização deste trabalho.

A meu orientador e amigo Prof. Dr. Cesar Camillo Augusto Teixeira pela sinceridade, paciência, orientação e por ter acreditado em mim desde meus primeiros trabalhos de iniciação científica.

A meus amigos Erick Melo, Caio Viel, Carlos Faganello, Renan Garcia, Gabriel Fernandes, Israel Aono, Roberto Fernandes, Chaud Cédrick, Nicole Alu Lagrange, Luiza Mota e Ana Paula Correia, que me proporcionaram conhecimento, sabedoria e momentos de descontração imprescindíveis.

A todos os membros do Laboratório LINCE que tiveram contribuição indispensável em diversas etapas deste trabalho.

"Sabei escutar, e podeis ter a certeza de que o silêncio produz, muitas vezes, o mesmo efeito que a ciência".

Napoleão Bonaparte

RESUMO

A evolução da Computação Gráfica apoiada no desenvolvimento computacional possibilita criações visuais cada vez mais avançadas promovendo benefícios a diversas áreas como: Artes, Arquitetura, Design de produto, Design visual, Jogos, Cinema, Engenharia, Geoprocessamento e Medicina. Porém, a capacidade de processamento exigida por aplicações que requerem elevado grau de realismo restringe o uso dessas aplicações a uma parcela pequena de toda a gama de dispositivos computacionais instalados e de uso da população. Recentemente, a evolução das tecnologias de rede móvel trouxe uma possível solução à incapacidade da maioria dos dispositivos móveis em lidar com aplicações interativas e de elevado grau de realismo. A solução baseia-se no processamento remoto das aplicações, em servidores com elevada capacidade de processamento, e na entrega das imagens produzidas para os dispositivos móveis feita através de sua conversão em vídeo e transmissão por *streaming*. Este trabalho explora uma solução de processamento remoto para interação com ambientes arquitetônicos através de tablets. Deu-se ênfase às questões de manutenção de elevado grau de realismo, dificultada pelas exigências de processamento em tempo real de aplicações interativas, à adaptação e combinação de ferramental disponível no mercado e à mitigação da latência através de recursos de interface com usuário. A avaliação dos resultados foi realizada em sessões de treinamento, interação e entrevistas com usuários voluntários.

Palavras-chave: computação gráfica, mídia de alto desempenho, interface remota, renderização remota, plataforma móvel, HPM

ABSTRACT

The evolution of Computer Graphics, supported by the computational development, enables increasingly advanced visual creations and promotes benefits to various areas such as: Art, Architecture, Product Design, Visual Design, Games, Movies, Engineering, GIS and Medicine. However, the processing power needed for applications that require a high degree of realism, restricts the use of these applications to a small portion of the entire range of the computing devices in use by the population. Recently, the evolution of mobile network technologies brought a possible solution to the inability of most mobile devices in dealing with interactive applications and a high degree of realism. The solution is based on the remote processing of applications on servers with high processing capacity, and delivery of the images produced for mobile devices made by its conversion into video and streaming broadcasts. This master dissertation explores a remote processing solution for interacting with architectural environments through tablets. Emphasis was placed in: keeping high degree of realism, difficult to achieve due to the processing requirements in real-time interactive applications; the adaptation and combination of tools available in the market; and in mitigating latency through different ways of user interaction. The evaluation of the results was carried out in training sessions, interaction and interviews with volunteers.

Keywords: graphics computing, high performance media, remote interface, remote rendering, mobile platform, HPM

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HPM – *High Performance Media*

MM-HPM – *Multimedia with High Performance Media*

NCL – *Nested Context Language*

SBTVD – Sistema Brasileiro de Televisão Digital

IHC – Interação Humano-Computador

QoE – *Quality of Experience*

QoS – *Quality of Service*

IDE - *Integrated Development Environment*

XML - *eXtensible Markup Language*

PNG - *Portable Network Graphics*

LINCE – Laboratório de Inovação em Computação e Engenharia

3D – *Three-dimensional*

3G – *Third Generation of Mobile Telecommunications Technology*

LTE – *Long-Term Evolution*

RAM – *Random Access Memory*

VRAM – *Video Random Access Memory*

GPU – *Graphics Processing Unit*

CPU – *Central Processing Unit*

API – *Application Programming Interface*

DFAO – *Distance Field Ambient Occlusion*

DFGI – *Distance Field Global Illumination*

VXGI – *Voxel Global Illumination*

FBX – *Filmbox*

HUD – *Heads-up Display*

AMQP – *Advanced Message Queuing Protocol*

QUA – *Queue Android to Unreal*

QAU – *Queue Unreal to Android*

ID – *Identity*

SO – Sistema Operacional

SUMÁRIO

Capítulo 1

INTRODUÇÃO

O objetivo deste capítulo é apresentar o contexto no qual se insere este trabalho bem como as motivações, objetivos e metodologia relacionados à sua concepção e ao seu desenvolvimento.

1.1 Contexto

A evolução da Computação Gráfica apoiada no desenvolvimento computacional possibilita criações visuais cada vez mais avançadas. Um computador *desktop* de última geração é capaz de processar um ambiente virtual altamente realista (PLANTE, 2015) (STUART, 2015). Esse tipo de aplicação caracteriza uma HPM. HPM, ou "*High Performance Media* é definida como mídia sintética interativa que necessita alto desempenho de processamento para ser produzida, que é consumida por seres humanos, e que promove a sensação de realismo, principalmente nos sentidos de visão e audição" (VIEL et al., 2012).

Com tais características, HPM potencialmente pode trazer benefícios a diversas áreas como: Artes, Arquitetura, Design de produto, Design visual, Jogos, Cinema, Engenharia, Geoprocessamento e Medicina. Porém, a capacidade de processamento exigido por uma HPM restringe o uso dessa mídia a uma parcela pequena de toda a gama de dispositivos computacionais disponíveis no mercado. Apesar do rápido desenvolvimento e popularização dos dispositivos portáteis e móveis, como *notebooks, tablets e smartphones*, a tendência é de que sempre exista um *gap* significativo, entre esses dispositivos e computadores *desktop* de última geração e também servidores especializados, em relação à capacidade de processamento gráfico (GARBER, 2013).

Recentemente, a evolução da Internet impulsionou o surgimento de um novo tipo de serviço na indústria de jogos eletrônicos, mercado que movimentou 79 bilhões

de dólares em 2012, chamado *Cloud Gaming*. Esse serviço baseia-se em uma solução de processamento remoto com *streaming* de vídeo para disponibilização de jogos *on demand* e é aposta de empresas importantes na área de entretenimento como NVIDIA (NVIDIA, 2015a), Amazon (AMAZON, 2015a) e Sony (MARTIN, 2015).

Em 2011 e 2012, membros do laboratório LINCE – DC/UFSCar fizeram testes com processamento remoto de HPM para televisores, dispositivos móveis e *set-top-box* envolvendo a linguagem NCL (GOMES SOARES et al., 2010) para possibilitar interatividade (VIEL et al., 2011) (VIEL et al., 2012).

1.2 Motivação e Objetivos

Áreas que utilizam computação gráfica para gerar mídias de alto realismo visual, e que possam ser reproduzidas em qualquer dispositivo, fazem uso do pré-processamento[1]. Os setores Imobiliário e Automotivo, por exemplo, desejam apresentar seus futuros produtos com alto realismo sem limitar a variedade de dispositivos que seus futuros clientes precisarão para visualizá-los. Então, recorrem ao pré-processamento para gerar imagens e vídeos, que não demandam alta capacidade de processamento para serem visualizados. Essa técnica, porém, não permite que se obtenha mídias com o mesmo grau de interatividade possibilitado pelo processamento em tempo real, como o encontrado em videogames, por exemplo.

Com o processamento remoto de HPM, limita-se a qualidade visual das imagens processadas em tempo real apenas à capacidade de *hardware* dos servidores remotos, o que torna os requisitos de *hardware* dos dispositivos clientes semelhante aos necessários para se reproduzir um vídeo via *streaming*. Surge então a possibilidade de se aliar a interatividade ao realismo visual e independência de *hardware* exigidos pelas áreas citadas acima.

[1] Modelos em 3D pré-processados são aqueles em que a geração das imagens ou vídeos não ocorre em tempo de execução. Por esse motivo, pode ser gasto o tempo que for necessário para se atingir o nível de realismo visual desejado. Por outro lado, o material gerado possui baixo potencial de interatividade, pois não possibilita alterações nas imagens que já foram geradas. Exemplos: Vídeos e imagens feitos com computação gráfica usados em marketing imobiliário e automotivo.

Os *Game Engines*[2], evoluíram de tal forma nos últimos anos que a qualidade visual de suas imagens se assemelham à das imagens com uso de pré-processamento, o que possibilita que a solução de HPM remota possa atender às exigências de realismo visual de áreas que ainda fazem uso do pré-processamento. (PLANTE, 2015). A Figura 1 mostra um exemplo de ambiente 3D pré-processado com os componentes de *software* Autodesk 3D Studio (AUTODESK, 2015a) e Vray (DYNAMICS, 2015). O tempo de processamento necessário para geração de uma imagem como essa é de cerca de quatro horas em um computador *desktop* de alto desempenho[3]. A Figura 2 mostra o mesmo ambiente, devidamente adaptado, e processado em tempo real no mesmo computador com o *Game Engine* Unreal 4 (EPIC GAMES, 2014). Os resultados finais são semelhantes, mas as produções de conteúdo para cada uma dessas abordagens diferem em diversos aspectos como limite de polígonos por cena, recursos de simulação de iluminação, simulação de física, etc. Isso pode dificultar a transição de áreas que fazem uso do pré-processamento para uma possível solução de processamento em tempo real com HPM remota. Porém, com técnicas adequadas de conversão de conteúdo e evolução constante dos *Game Engines* (RABELLO, 2013), essa dificuldade não inviabiliza a transição.

[2] *Game Engine* é um programa ou conjunto de bibliotecas para auxiliar no desenvolvimento de jogos eletrônicos ou outras aplicações com gráficos 3D ou 2D processados em tempo real.

[3] Configuração: Processador Intel Core i7, 16gb de RAM e placa de vídeo NVIDIA GTX980Ti 6gb de VRAM

Figura 1 - Exemplo de visualização arquitetônica pré-processada (Fonte: Site da Evermotion - http://www.evermotion.org/shop/show_product/archinteriors-vol-29/9520)

Figura 2 - Exemplo de visualização arquitetônica processada em tempo real (Fonte: Próprio autor)

Soluções de processamento remoto estão sujeitas a latência oriunda de diversos fatores como a distância física entre os componentes da arquitetura, complexidade e qualidade da rede utilizada para comunicação entre os componentes e tempo de processamento das informações no cliente e no servidor. Como a HPM possui interatividade em tempo real, a latência pode prejudicar a qualidade da experiência do usuário se não for suficientemente pequena. Esse é um problema real na viabilização de aplicações interativas processadas remotamente, incluindo o *Cloud Gaming* (SHEA; NGAI, 2013), e existem diversos trabalhos que estudam e buscam formas de combater as causas e os efeitos da latência em diferentes componentes da arquitetura. Pode-se, por exemplo, organizar a rede de forma que os usuários estejam sempre fisicamente próximos o suficiente dos servidores(CHOY et al., 2014); reduzir a latência da solução através da otimização dos algoritmos de codificação e decodificação do vídeo transmitido por *streaming* (GODOY, 2014)*;* utilizar técnicas de predição para antecipar as ações do usuário e transmitir partes do vídeo com antecedência (LEE et al., 2015); ou mesmo substituir o *streaming* de vídeo por outras técnicas como imagens panorâmicas, que ficam armazenadas no dispositivo e permitem o processamento local instantâneo para determinados tipos de interação (GHILETIUC; FÄRBER; BRÜDERLIN, 2013).

Porém, substituições do *streaming* de vídeo por outras abordagens estão propensas a exigir o desenvolvimento de novos padrões e ferramentas de codificação e decodificação. O emprego de capacidades de predições avançadas pode requerer recursos inexistentes nos principais *Game Engines* utilizadas atualmente. Abordagens de combate à latência que requerem a alteração de elementos adotados pelas principais soluções de *Cloud Gaming* podem dificultar a adoção desse formato.

Neste trabalho, é proposta, implementada e explorada uma solução de processamento remoto para interação com ambientes arquitetônicos 3D através de *tablets*. Deu-se ênfase às questões de manutenção de elevado grau de realismo, dificultada pelas exigências de processamento em tempo real de aplicações interativas, à adaptação e combinação de ferramental disponível no mercado e à mitigação da latência através de diferentes soluções de interface com usuário. A avaliação dos resultados foi realizada em sessões de treinamento, interação e entrevistas com usuários voluntários.

Com um ambiente arquitetônico, busca-se representar aplicações 3D que necessitam de realismo e possuem as características típicas de uma HPM. O emprego de *tablets* foi feito para se representar a parcela de dispositivos móveis que tem maior potencial para consumo de aplicações com as caraterísticas da HPM. A busca pelo uso de ferramentas de arquitetura de processamento remoto disponíveis no mercado visa a maior compatibilidade da solução explorada com soluções de *Cloud Gaming* consolidadas.

1.3 Metodologia

Foram utilizados e experimentados conceitos e tecnologias através do desenvolvimento modular de dois artefatos de *software* com duas versões cada: Um aplicativo 3D de alto realismo compatível com as definições de HPM, a ser processado em um servidor, e um aplicativo de dispositivo móvel a ser processado em um cliente. Ambos foram integrados em duas versões de uma solução de processamento remoto, baseada em *streaming* de vídeo, formada por elementos de *software* e *hardware* estudados e selecionados dentre opções disponíveis no mercado. A primeira versão, denominada Solução Tradicional, foi construída utilizando-se formas de interação tradicionais, sem a preocupação com a latência. Já na segunda versão, denominada Solução Alternativa, foram estudadas e desenvolvidas formas de interação que pudessem mitigar a latência e promover melhor qualidade da experiência.

O levantamento bibliográfico realizado ajudou na seleção de técnicas e componentes de *software* e na concepção de formas de interação. Para avaliação da solução, foram comparados e analisados os resultados de testes subjetivos e entrevistas com usuários que utilizaram as duas versões da solução sujeitas a latência e opinaram sobre a qualidade da imagem e funcionalidades da solução.

1.4 Organização do Texto

No Capítulo 2 - são apresentados fundamentos e conceitos importantes no contexto deste trabalho. No Capítulo 3 - são abordados alguns trabalhos relacionados que exploram formas de se reduzir a latência ou seus efeitos em aplicações 3D que utilizam arquiteturas de processamento remoto. O Capítulo 4 - é dedicado ao desenvolvimento do trabalho. Nele são relatados os processos de decisão e implementação de todos os componentes da solução proposta. A elaboração da avaliação, realização dos testes e análise dos resultados são relatados no Capítulo 5 - Por fim, a Conclusão do trabalho é delineada no Capítulo 6 -

Capítulo 2

FUNDAMENTOS E CONCEITOS RELEVANTES

O objetivo deste capítulo é apresentar os principais conceitos utilizados neste trabalho e como eles se encaixam no projeto. São eles HPM, o tipo de mídia utilizada no projeto; Arquitetura de Processamento Remoto e *Cloud Gaming*, soluções utilizadas para processamento da HPM; e Latência em jogos, Usabilidade e Qualidade da Experiência, conceitos de apoio na caracterização dos efeitos da latência da arquitetura para o usuário e formas de mitiga-los através da interface de usuário.

2.1 HPM – *High Performance Media*

HPM, ou *High Performance Media*, é uma mídia sintética interativa que necessita alto desempenho de processamento para ser produzida, que é consumida por seres humanos, e que promove a sensação de realismo, principalmente nos sentidos de visão e audição (VIEL et al., 2012) . Como "alto desempenho" não possui uma delimitação precisa e é um conceito temporal, a definição de HPM também passa a depender do momento tecnológico. Toma-se a capacidade usual e atual de processamento de equipamentos eletrônicos *commodity* como limite inferior de "alto desempenho" na definição de HPM.

Devido a sua natureza portátil, dispositivos móveis como *smartphones* e *tablets* possuem limitações físicas, como tamanho e disponibilidade de energia, que os colocam em desvantagem frente a computadores *desktop* e servidores que possuem

espaço para empregar soluções robustas de dissipação térmica. Com isso, esses últimos podem fazer uso de processadores mais poderosos do que os dispositivos menores e dependentes de baterias. A tendência é de que sempre exista diferença de desempenho entre esses dois tipos de dispositivos (GARBER, 2013). Isso faz com que a HPM não possa ser processada localmente em dispositivos móveis, pois, por definição, ela necessita dos equipamentos *commodity* de melhor desempenho disponíveis no momento tecnológico.

Jogos eletrônicos com gráficos realistas, que fazem uso do potencial de processamento de computadores *desktop* que se encontram no estado da arte, são exemplos típicos de HPM.

2.2 Arquitetura de Processamento Remoto

"Sistema de computação remoto é um sistema de processamento de dados com seus terminais distantes da unidade central de processamento, que podem ser usados por usuários para compilação, depuração, teste e execução de programas na unidade central" (LAPEDES, 2002, tradução nossa). Neste trabalho, considera-se Arquitetura de Processamento Remoto como um sistema de computação remoto formado por três componentes básicos: Servidor, Cliente e Gerenciador de Comunicação ou *broker*. O servidor refere-se à unidade central de processamento enquanto e o Cliente ao refere-se ao terminal. O Gerenciador de Comunicação é um componente opcional que intermedeia a troca de mensagens entre servidor e cliente. No contexto de HPM e outras mídias 3D interativas, delega-se funções mais específicas a cada um desses componentes como descrito abaixo:

- **Servidor Remoto de Processamento da HPM**: Responsável por gerenciar e processar a HPM, que é constituída tipicamente de cenas complexas em 3D geradas em tempo real por um *Game Engine.* Esse componente também codifica as informações geradas pelo *Game Engine*, com uso de técnicas específicas, e as envia para o Cliente. Neste trabalho, a técnica utilizada para esse fim é a de *streaming* de vídeo, no qual os quadros de imagem gerados pelo *Game Engine* são codificados em um formato de vídeo.

- **Gerenciador de Comunicação:** Trata-se de um *broker,* responsável por gerenciar as mensagens trocadas entre servidor e cliente. Esse componente não é essencial para o funcionamento da Arquitetura de Processamento Remoto. Caso ele não seja usado, o gerenciamento da troca de mensagens fica a cargo dos próprios cliente e servidor. O uso do *broker* pode facilitar a criação e manutenção de soluções de processamento remoto mais complexas, com diversos tipos de clientes e servidores, além de contribuir com a modularidade da arquitetura.
- **Dispositivo Cliente:** Contém o *software* que fornece a visualização do *streaming* de vídeo e possibilita a interação do usuário com a HPM através da captura de seus comandos e envio de mensagens para o servidor através do Gerenciador de Comunicação.

A Figura 3 ilustra a arquitetura acima e seus componentes.

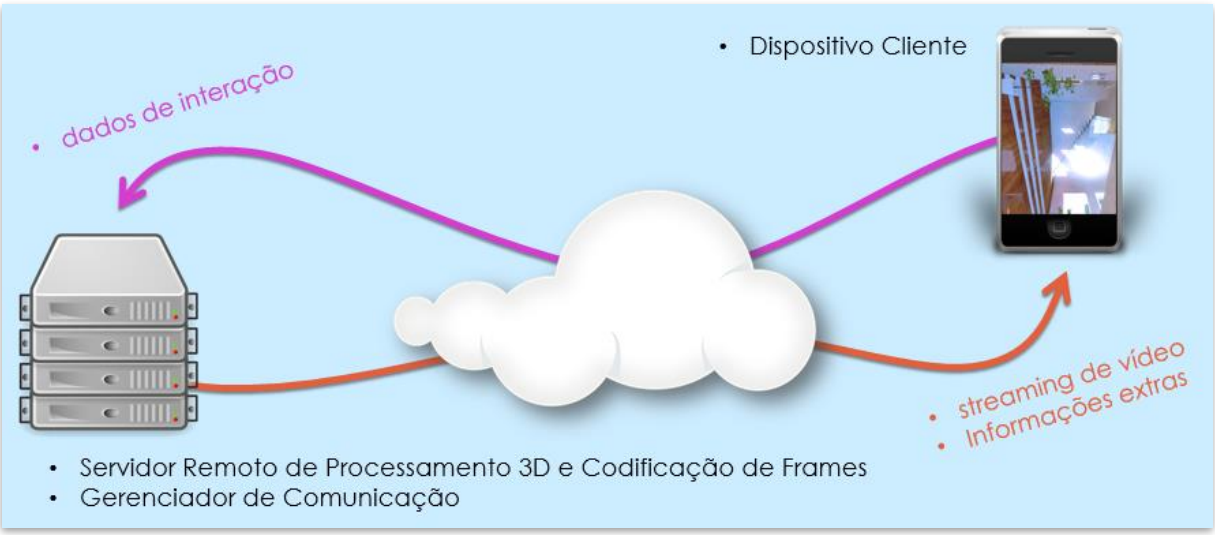

Figura 3 - Arquitetura de Processamento Remoto de HPM. (Fonte: Próprio autor)

No Servidor Remoto de Processamento de HPM, uma HPM é processada em tempo o real por um *Game Engine*. Toda a lógica da interatividade com o ambiente 3D ocorre dentro dessa aplicação, que tem suas imagens capturadas e codificadas em um formato de *streaming* de vídeo a ser enviado para o Dispositivo Cliente, responsável por decodifica-lo. A HPM também é responsável pelo envio de mensagens de sincronismo caso o Dispositivo Cliente combine elementos processados localmente com o vídeo recebido do servidor.

Enquanto assiste ao vídeo, o usuário pode interagir com a Aplicação, que está preparada para receber as interações planejadas. Os dados de entrada do usuário são então enviados para a aplicação no servidor, que realiza as atualizações adequadas no ambiente 3D e gera novas sequencias de imagens a serem enviadas para o cliente em forma de *streaming* de vídeo. As mensagens trocadas entre servidor e cliente podem ser intermediadas e gerenciadas por um Gerenciador de Comunicação. Esse ciclo se repete de forma que a sensação causada no usuário seja a de que a HPM está sendo processada no próprio dispositivo com o qual ele interage.

Cada um dos componentes citados contribui com uma parcela da latência total da arquitetura. Essa latência é um fator crítico na experiência do usuário, pois, se não for suficientemente pequena, pode causar um atraso perceptível entre as interações do usuário e as atualizações correspondentes das imagens exibidas na tela.

Em 2012, o percentil 95 de latência para conexões 3G, LTE e Wi-Fi em áreas metropolitanas dos Estados Unidos passam de 600ms, 400ms e 300ms respectivamente (SOMMERS; BARFORD, 2012) (HUANG et al., 2012). Essas latências são frequentemente causadas pela última milha da rede. Conexões com servidores distantes podem causar resultar em valores ainda maiores. À latência da rede ainda é somado o tempo de processamento da aplicação 3D em si, geração do *streaming* de vídeo e decodificação do mesmo no dispositivo cliente. No serviço de *Cloud Gaming* Gaikai (NVIDIA, 2012), que utiliza soluções da Nvidia, esses processos chegam a mais de 100 milissegundos . Em um cenário real, utilizando-se de serviços já disponíveis no mercado, a latência média da solução proposta com processamento remoto pode passar de 500 milissegundos em conexões Wi-Fi e 800 milissegundos em conexões 3G.

2.3 Cloud Gaming

O avanço tecnológico na área de computação em nuvem permitiu o recente surgimento de um serviço de processamento de jogos eletrônicos de forma remota chamado *Cloud Gaming* (SHEA; NGAI, 2013). Nesse tipo de serviço, os jogos deixam de ser processados localmente no computador, *tablet*, *smartphone* ou videogame do usuário e passam a ser processados em servidores especializados. O dispositivo do

usuário funciona então apenas como um terminal que mostra as imagens geradas pelo jogo no servidor que são recebidas através da internet em forma de *streaming* de vídeo. As interações do jogador com o dispositivo são enviadas para o jogo no servidor, que as intenderá da mesma forma que ocorreria em uma situação de processamento local. Em uma analogia simplificada, é como se os cabos que ligam a TV e um *joystick* ao videogame fossem substituídos por uma conexão com a internet, possibilitando que o videogame esteja em qualquer lugar do mundo.

O *Cloud Gaming* permite que os jogos sejam comercializados como um serviço do tipo *on demand*, semelhante ao que ocorre com vídeos em plataformas como o YouTube e Netflix. Esse tipo de formato traz diversas vantagens para a indústria de jogos se comparadas ao formato tradicional de venda de jogos por unidade, dentre elas:

- **Independência do *hardware*:** Os jogos não dependem da capacidade de processamento do dispositivo do usuário, mas apenas da capacidade do servidor. Isso possibilita que qualidade gráfica tire proveito do que há de melhor em *hardware* sem a exigência de que o usuário adquira um equipamento com a capacidade necessária para tanto.
- **Facilidade de manutenção:** Os jogos podem ser corrigidos e atualizados sem a necessidade de que o usuário faça o download de arquivos de atualização e os instale.
- **Acesso instantâneo:** O usuário não precisa esperar pelo download e instalação de um jogo para começar a utilizá-lo, pois o acesso ao serviço *on demand* é instantâneo.
- **Combate à pirataria e uso indevido:** O usuário deixa de possuir uma cópia física ou digital do *software* do jogo. Isso elimina problemas com pirataria ou uso indevido do *software* como revenda ou aluguel sem autorização, por exemplo.

Grandes empresas como Amazon, Sony e Nvidia estão investindo em *Cloud Gaming* e acreditam que esse seja o futuro do mercado de videogames. A Gaikai (GAIKAI, 2015), uma das pioneiras a acreditar nesse formato em 2008 foi recentemente adquirida pela Sony (MARTIN, 2015) para levar o serviço de *Cloud*

Gaming para seu console Playstation 4 como forma de fornecer acesso a jogos incompatíveis com o *hardware* desse aparelho.

Por se tratar de um serviço que depende da Internet, o *Cloud Gaming* está sujeito a problemas relacionados à rede como quedas de conexão, *jitter*, perda de pacotes e latência. Pela sua natureza interativa, jogos são aplicações extremamente sensíveis à latência, o que impossibilita o uso de *buffers* para combater inconsistências na qualidade da rede, por exemplo. Esses problemas motivam diversos trabalhos que buscam soluções de combate à latência e seus efeitos em serviços de jogos processados remotamente como o *Cloud Gaming.* Alguns desses trabalhos são apresentados no item 2.4 e no Capítulo 3 -

2.4 Latência em Jogos Multiplayer e Cloud Gaming

O impacto da latência em interfaces de usuário que dependem da rede não é um problema recente. Em 2002, Pantel e Wolf (PANTEL; WOLF, 2002) realizaram um trabalho para detectar o impacto que a latência da rede poderia causar na experiência do usuário com jogos *multiplayer*. Foram selecionados dois jogos comerciais de corrida e uma série de testes, inclusive subjetivos, foram realizados. Na arquitetura *multiplayer* utilizada, o atraso da rede impacta na experiência de todos os jogadores mostrados na tela, incluindo o jogador local. Isso ocorre para garantir que os eventos se mantenham sincronizados em todos os dispositivos envolvidos no jogo. Se o jogador local movimenta seu carro para a direita e essa informação demora um segundo para atingir outros jogadores, o carro do próprio jogador também demorará um segundo para responder ao seu comando. Ao fim dos testes, o autor conclui que latências acima de 100 milissegundos devem ser evitadas em jogos de corrida. Também é considerado que esse tipo de jogo é o mais sensível à latência e jogos mais lentos podem tolerar latências maiores.

Pesquisas com jogos locais com *multiplayer online* indicaram que atrasos acima de 60 milissegundos já são percebidos pelo usuário em jogos com visão em primeira pessoa (QUAX et al., 2004), e 100 milissegundos são o suficiente para causar incômodo (BEIGBEDER et al., 2004) (DICK; WELLNITZ; WOLF, 2005). Em um teste

com usuários de um jogo de estratégia online detectou-se que, enquanto a média de horas seguidas de jogatina se mantém em 4 horas em um cenário com 150 milissegundos de latência, 250 milissegundos diminuem o engajamento dos jogadores para menos de 1 hora (CHEN; HUANG; LEI, 2006).

Há diferenças importantes entre um jogo processado totalmente de forma remota, como no *Cloud Gaming*, e um jogo processado localmente com alguns recursos *multiplayer* via rede. Enquanto que esse último troca com o servidor apenas informações lógicas do status dos jogadores, o primeiro troca informações lógicas e recebe *streaming* de vídeo de alta resolução com importantes restrições de latência. Isso eleva em muito os requisitos de QoS da rede e sua importância frente à QoE da aplicação (JARSCHEL et al., 2011). Motivado por essa problemática, Jarschel et al conduziu um estudo subjetivo sobre a percepção dos usuários em relação à qualidade da experiência de jogos processados remotamente (JARSCHEL et al., 2011). Foi desenvolvido um ambiente de testes que emula esse tipo de serviço, com testes específicos para determinar a QoE. Diversas configurações simularam diferentes situações de um sistema de *Cloud Gaming* como variações de latência e perda de pacotes. Aplicando-se os testes em usuários, obteve-se resultados que possibilitaram conclusões gerais sobre o impacto dos diferentes parâmetros de QoS testados sobre a QoE do sistema.

Para os testes, foram disponibilizados três diferentes tipos de jogos dentre os quais os participantes poderiam escolher um ou mais para testar. Antes do início dos testes, o jogador deveria dizer qual seu tipo de jogo preferido e quais são os seus níveis de habilidade em cada tipo que escolheu. Os testes então começariam com o primeiro minuto de forma perfeita, isto é, sem a simulação de nenhum problema na rede. Após esse período, os cenários de diferentes parâmetros de QoS se iniciariam em ordem aleatória. Ao final de cada cenário, o jogador seria questionado sobre a qualidade da imagem e da fluidez do jogo sob parâmetros de 1 a 5, de ruim a excelente. Ao final de todos os testes, o participante responderia se estava disposto a pagar por um serviço que fornecesse uma experiência como aquela que acabou de ter, para qualquer jogo que quisesse. Essa pergunta buscava obter uma impressão geral dos testes por parte dos participantes.

Após a devida avaliação dos resultados, Jarschel et al conclui que a experiência proporcionada pelo *Cloud Gaming* não depende apenas dos parâmetros de QoS como latência e perda de pacotes, mas também do contexto da aplicação. Em jogos que

priorizam os efeitos visuais, por exemplo, a qualidade do *streaming* de vídeo tem um papel mais importante do que em jogos com gráficos mais simples. Da mesma forma, os jogos com mais ação são mais sensíveis à latência do que os que possuem jogabilidade mais lenta.

Ainda que jogos de estilos diferentes prejudiquem a performance do jogador com intensidades diferentes quando sujeitos à mesma latência, jogos em *Cloud Gaming* são em média mais sensíveis do que jogos locais com *multiplayer online* nesse aspecto. Um estudo feito por Mark Claypool e David Finkel em 2014 (CLAYPOOL; FINKEL, 2014) mostrou que 100 milissegundos é o suficiente para começar a impactar na performance dos usuários em qualquer estilo de jogo em *Cloud Gaming.* Em jogos locais com *multiplayer online*, esse tempo de latência afeta apenas jogos de tiro em primeira pessoa. Os autores supõem que a causa desses resultados seja o tempo de resposta a todas as interações do usuário, que é mais alto em jogos de *Cloud Gaming*. Nos jogos locais com *multiplayer online,* elementos do jogo que dependem da rede estão sujeitos a atrasos e isso pode prejudicar a jogabilidade. Porém, as respostas aos comandos do usuário para movimentação de seu próprio personagem geralmente são imediatas. Nos jogos em *Cloud Gaming*, todos os aspectos do jogo estão sujeitos aos atrasos, inclusive os comandos de seu próprio personagem.

2.5 Usabilidade e Qualidade da Experiência

O conceito de HPM compartilha muitas características com jogos eletrônicos. Principalmente jogos eletrônicos em 3D com gráficos de última geração. Devido a essa semelhança, estudos de Usabilidade e Qualidade da Experiência em jogos podem contribuir com este trabalho. As sessões a seguir abordam esses dois conceitos e analisa como a latência interfere na usabilidade a ponto de afetar a qualidade da experiência.

2.5.1 Usabilidade em Interfaces de *Software* e Jogos

No campo da computação, Nielsen (NIELSEN, 1994a) e Shneiderman (SHNEIDERMAN, 1986), importantes nomes na área de Usabilidade, definem o conceito de usabilidade basicamente pelos seguintes componentes:

- **Aprendizado**: Quão fácil é para o usuário realizar suas tarefas na primeira vez que utiliza a interface.
- **Eficiência**: Após aprender a utilizar a interface, quão rápido o usuário consegue realizar suas tarefas.
- **Erros**: Quantos e quão severos são os erros que o usuário comete com a interface e qual a facilidade de se recuperar.
- **Memorização:** Qual a facilidade de se lembrar do uso da interface após certo tempo sem utilizá-la.
- **Satisfação:** Quão prazerosa é a utilização da interface.

Nos primórdios da computação, os computadores eram tão caros e restritos que fazia sentido exigir um alto grau de treinamento para a sua utilização com máxima eficiência. Porém, com a popularização do computador pessoal e a redução do seu custo, dedicar parte do poder de processamento a uma interface que facilitasse a vida do usuário passou a ser uma prática economicamente interessante. Nesse momento o conceito de usabilidade em *software* passou a ter maior importância (NIELSEN, 1994a).

Os videogames também contribuíram consideravelmente para o aumento da preocupação com a usabilidade das interfaces de usuário em *software*. Os jogos eletrônicos deixavam cada vez mais claro que era possível ter uma experiência agradável com o uso de um computador, e isso aumentava o nível de exigência do usuário. Em 1982, antes mesmo da área de IHC receber a contribuição de Nielsen com o método de inspeção e heurísticas voltadas para a usabilidade de *software* (NIELSEN; MOLICH, 1990), Thomas W. Malone, do Xerox Palo Alto Research Center, foi motivado a realizar um trabalho em que propôs um *framework* que possibilitava a criação de interfaces mais agradáveis. Ele realizou alguns estudos para entender quais características dos jogos eletrônicos eram responsáveis pela diversão que causava aos usuários e, com base nessas características, desenvolveu heurísticas para guiar o design de qualquer programa de computador (MALONE, 1982).

Posteriormente, suas contribuições acabaram servindo de base para a maioria dos trabalhos sobre usabilidade em jogos eletrônicos. Algumas de suas heurísticas como "Possuir um objetivo claro e fornecer *feedback* ao usuário sobre o quão perto ele está do objetivo" e "Permitir o ajuste gradativo do nível de dificuldade" permanecem entre as principais heurísticas para jogos e *software* nos trabalhos mais atuais.

Definidas as características que determinam se uma interface tem ou não boa usabilidade, o desafio passou a ser o desenvolvimento de métodos e heurísticas que garantissem o projeto de interfaces com boa usabilidade. Uma série de trabalhos foram desenvolvidos em busca desses métodos e de quando e como aplica-los no desenvolvimento de *software*, que agora passa a reconhecer a importância do usuário. Dentre os principais, encontra-se o conjunto de heurísticas desenvolvidas por Nielsen e Molich em 1990 (NIELSEN; MOLICH, 1990) (MOLICH; NIELSEN, 1990), posteriormente refinado por Nielsen com a análise de 249 problemas de usabilidade (NIELSEN, 1994b) resultando em seu livro Heuristic Evaluation (NIELSEN, 1994a). Segundo esse trabalho, os 10 princípios básicos a serem seguidos em busca de uma interface com boa usabilidade são:

- **Visibilidade do estado do sistema**: O sistema deve manter o usuário devidamente informado sobre o que está acontecendo;
- **Semelhança com o mundo real**: O sistema deve utilizar a linguagem do usuário ao invés de termos estritamente relacionados ao próprio sistema;
- **Controle e liberdade**: O usuário constantemente realiza operações equivocadas e precisa de uma maneira clara de desfazê-las;
- **Consistência e padrões**: O usuário pode ser confundido por palavras iguais usadas em diferentes situações e com diferentes significados. Deve-se seguir uma plataforma de convenções;
- **Prevenção de erro**: Melhor do que a possibilidade de corrigir um erro é o sistema que evita que o erro aconteça ou fornece opções para evita-lo;
- **Reconhecimento ao invés de recordação**: Ao invés de exigir que o usuário se lembre de informações de etapas anteriores, o sistema deve sempre fornecer formas simples de acessar tais informações a qualquer momento;

- **Flexibilidade e eficiência**: O sistema deve fornecer recursos que facilite ainda mais a vida do usuário experiente;
- **Design minimalista**: O sistema não deve prover informação irrelevante ou raramente necessária.
- **Ajuda ao usuário no reconhecimento, diagnóstico e recuperação de erros**: As mensagens de erros devem ser simples e claras, indicando precisamente o problema e sugerindo uma solução de forma construtiva.
- **Ajuda e documentação**: É melhor que o sistema seja autoexplicativo, mas quando necessária, a documentação de ajuda deve ser construtiva e de fácil pesquisa.

Oriundo dos problemas de usabilidade detectados em diversos testes de interfaces, esse conjunto de heurísticas busca guiar o design do sistema para a interface ideal do ponto de vista do usuário. Ele também passou a servir de base para o desenvolvimento de heurísticas especializadas em diversos trabalhos posteriores.

Se uma aplicação está sujeita a latência que afeta a interface de usuário, ou seja, causa atrasos nas respostas para as ações do usuário, isso pode ferir o primeiro dos 10 princípios básicos de Nielsen: Visibilidade do estado do sistema. Durante o período de espera causado pela latência, a interface deve deixar claro para o usuário que ele deve esperar.

2.5.2 QoE - Qualidade da Experiência

O termo “Qualidade da Experiência” (QoE – *Quality of Experience*) começou a ser comumente mencionado em trabalhos relacionados a web e serviços de rede no final da década de 90 (SILLER; WOODS, 2003). Apesar de diferentes definições serem usadas na literatura, no geral esse termo se relaciona à medição da qualidade de um sistema do ponto de vista do usuário, remetendo a um dos elementos de usabilidade anteriormente mencionados: Satisfação do usuário. Patrick et al (PATRICK et al., 2004) define QoE como as características das sensações, percepções e opiniões das pessoas sobre o sistema. Essas características podem ser de agradáveis a frustrantes.

A interface de um sistema ganha grande importância quando se fala em qualidade da experiência, pois é unicamente através dela que o usuário tem contato com esse sistema. Do ponto de vista do usuário, a interface é o próprio sistema, e

tudo que reflete no funcionamento dessa interface consequentemente reflete na qualidade da experiência do sistema como um todo. O termo “Qualidade de Serviço” (QoS – *Quality of Service*), muito utilizado na área de redes, se traduz na medição de características da rede como taxa de transmissão de dados, taxa de erros e outras que podem ser medidas e melhoradas. Ao tratar esse termo de forma mais abrangente, levando-o da rede para todo o sistema, pode-se dizer que a origem da medida de QoE é o que antes era considerado como a percepção do usuário sobre a QoS (SILLER; WOODS, 2003).

Neste trabalho são realizados testes com usuários a fim de se medir, dentre outros aspectos, a Qualidade da Experiência de duas opções de interface com usuário implementadas em situações de latência.

Capítulo 3

TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos que exploram soluções de processamento remoto para jogos com foco na mitigação dos efeitos da latência na experiência do usuário. Alguns exploram alguns conceitos semelhantes aos utilizados em algumas etapas do desenvolvimento deste trabalho.

Com o recente crescimento do *Cloud Gaming,* surgiram muitos estudos sobre os efeitos da latência na experiência do usuário em diferentes tipos de jogos que possam ser utilizados com processamento remoto. Há também diversos trabalhos sobre possíveis soluções da latência e seus efeitos nesse tipo de arquitetura.

Alguns conceitos utilizados neste trabalho, que se baseiam em processar alguns elementos da aplicação diretamente no dispositivo cliente, assemelham-se a conceitos utilizados em parte da solução de Kyungmin Lee et al. Sua proposta é a redução dos efeitos da latência em *Cloud Gaming* através de especulação (LEE et al., 2015), com uma série de técnicas para predição de possíveis ações do usuário e consequências que elas podem causar na aplicação 3D.

O *streaming* de vídeo enviado contém um conjunto de quadros extras além dos quadros principais a serem mostrados para o cliente. Os quadros extras possuem imagens de ações previstas pelo servidor e o dispositivo cliente é responsável por determinar localmente quais quadros deve mostrar, com base nas ações do usuário. Essas ações são categorizadas em dois tipos: Navegação, que consiste da rotação e translação e causa mudanças no campo visual do jogador; E Impulso, que consiste de eventos esporádicos como o acionamento de um objeto ou disparo de uma arma.

Para predição de ações do tipo Impulso, utiliza-se várias instâncias da aplicação 3D processadas de forma paralela no servidor. Cada uma das instâncias é

responsável por uma possível ação do usuário. Ações que naturalmente exigem um tempo de espera maior do que a latência da rede, como a troca de uma arma, por exemplo, não são previstas. Como consequência, ações que já exigem espera do usuário demorarão um pouco mais para se concretizar.

Para predição da Navegação foi desenvolvida uma técnica com base em Cadeias de Markov. Os movimentos do jogador pelo cenário são antecipados a partir de seus movimentos anteriores. Diferentemente das ações de Impulso, que simplesmente ocorrem ou não ocorrem, as ações de Navegação podem ocorrer de muitas maneiras diferentes. Translações podem ocorrer para qualquer direção e rotações de câmera podem ter qualquer ângulo. Com uma quantidade limitada de quadros extras no *streaming*, caso as predições não sejam precisas, o usuário notará que seu personagem se move para direções diferentes. Como solução desse problema, o autor utiliza uma técnica de vídeo panorâmico. Ao invés de enviar ao cliente um vídeo com área correspondente à tela do dispositivo, é enviado um panorama com áreas extras em todas as direções adjacentes à visão do jogador. Além disso, são enviadas informações de profundidade do ambiente 3D para que o dispositivo cliente possa realizar correções de perspectiva e oclusão de objetos conforme altera localmente o ângulo e translação do vídeo panorâmico. Dessa forma, as ações de Navegação do usuário se traduzem instantaneamente na manipulação do vídeo panorâmico de forma que as imagens se aproximem com o que seria gerado diretamente pela aplicação 3D. Se ainda houver divergência, um algoritmo de correção interpola as imagens corretas com o que foi gerado localmente, o que pode produzir pequenos artefatos visuais.

A solução de Kyungmin Lee et al é capaz de reduzir significativamente os efeitos de latências acima de 128 milissegundos quando comparada com a solução de processamento remoto com *streaming* de vídeo padrão, com resultados comparáveis a uma aplicação processada localmente. Porém, ela necessita de modificações extremas na aplicação 3D para introdução dos recursos de predição, que não são oferecidos pelos principais *Game Engines*. O formato de *streaming* de vídeo é específico, capaz de suportar quadros extras das predições, e necessita de codificador e decodificador especializados. Além disso, o consumo de banda do *streaming* de vídeo com predições para 128 milissegundos é 1,97 vezes maior do que o padrão. Esse valor aumenta ainda mais para predições de tempos maiores, o que

pode tornar a solução inviável em redes com restrições de banda ou consumo de dados, como em conexões de telefonia móvel.

Em (DIEPSTRATEN; GORKE; ERTL, 2004) é apresentada uma solução de processamento remoto com o envio de informações vetoriais em vez de se enviar *streaming* de vídeo. No servidor, ao invés dos gráficos da aplicação serem codificados em um vídeo formado por quadros, são convertidos em um conjunto de gráficos vetoriais descritos por uma quantidade de dados reduzida. Esses dados são enviados através da rede até o dispositivo cliente que interpreta a informação e desenhará a imagem na tela. Com o tráfego de dados reduzido, melhora-se o desempenho em redes com banda muito restrita para o tráfego de *streaming* de vídeo. Essa técnica é útil para mídias com gráficos simplificados, com poucos contornos, mas a quantidade de informação necessária para se representar todos os detalhes de uma imagem foto-realista de uma HPM pode ultrapassar a quantidade de dados da mesma imagem representada por um quadro de vídeo comum. Além disso, quanto mais detalhes em uma imagem vetorial, maior a capacidade de processamento necessário para decodificá-la no dispositivo cliente.

Há propostas de combate aos efeitos da latência no processamento remoto através de alternativas mais radicais ao *streaming* de vídeo padrão do *Cloud Gaming*. Em (GHILETIUC; FÄRBER; BRÜDERLIN, 2013) os autores propõem uma solução que distribui o processamento do ambiente virtual em 3D entre o servidor e o cliente. Ao invés de um vídeo, a aplicação no servidor gera uma série de imagens panorâmicas, que são utilizadas em um segundo ambiente virtual simplificado, a serem processadas no dispositivo cliente. Esse ambiente é composto por esferas concêntricas nas quais as imagens panorâmicas com regiões transparentes são aplicadas para representar objetos em diferentes distâncias do observador. A vantagem dessa solução é que as imagens panorâmicas não precisam ser atualizadas se o usuário navegar pelo cenário dentro de um certo limite, pois a disposição das esferas garante a sensação de profundidade do ambiente representado. Dessa forma, o fluxo de dados entre o servidor e o cliente é reduzido. Porém, em situações de movimentação extrema do observador ou do próprio cenário, característica de uma aplicação interativa como a HPM, serão necessárias atualizações frequentes das imagens panorâmicas. Por serem maiores do que frames comuns de vídeo, a transmissão dessas imagens pode resultar em um tráfego de dados ainda maior do que o utilizado por um *streaming* de vídeo. Além disso, atualizações frequentes do

ambiente simplificado no dispositivo cliente podem exigir mais poder de processamento do que ele é capaz de fornecer.

Em (TELER; LISCHINSKI, 2001) é feita uma proposta semelhante, na qual o ambiente virtual simplificado no dispositivo cliente é uma versão do ambiente original. A versão compõe-se de poucos polígonos, que são carregados gradualmente. Essa abordagem diminui a carga de processamento dos servidores, pelo fato de serem esses utilizados apenas para as simulações de interação e parte do processamento gráfico, que é finalizado nos dispositivos clientes. A qualidade gráfica, porém, é geralmente prejudicada por depender do poder de processamento gráfico do cliente.

Capítulo 4

DESENVOLVIMENTO DO TRABALHO

O objetivo deste capítulo é detalhar as etapas de desenvolvimento do trabalho com seus principais processos de decisões e implementações. Em 4.1 é descrito o processo de definição da arquitetura de processamento remoto adotada, bem como as ferramentas utilizadas em cada um de seus componentes. No item 4.2 são apresentadas as duas soluções de interface desenvolvidas neste trabalho e justificados seus propósitos. Em 4.3 são detalhados os processos de definição das formas de interação das duas soluções de interface. Em 4.4 é detalhado o desenvolvimento da HPM, o que inclui a escolha de ferramentas, uso de técnicas de modelagem 3D e implementação de formas de comunicação e gameplay. Por fim, em 4.5 é delineado o desenvolvimento da Aplicação Cliente, incluindo-se a escolha da plataforma móvel e implementação de formas de comunicação e funcionalidade de streaming.

4.1 Definição da Arquitetura de Processamento Remoto de HPM

Há diversas formas de se implementar uma arquitetura de processamento remoto de aplicações 3D que seja capaz de lidar com os efeitos da latência. Muitos trabalhos, já citados nos capítulos anteriores, propõe alternativas ao *streaming* de vídeo, padrão usado pelas principais soluções atuais de *Cloud Gaming*, e funcionalidades específicas da aplicação 3D que não são oferecidas pelos principais *Game Engines* disponíveis no mercado. Com esses requisitos, essas soluções dificultam a possibilidade de uso das soluções de *Cloud Gaming* já consolidadas no

mercado como o Amazon AppStream (AMAZON, 2015b), Otoy X.IO (OTOY, 2015), Nvidia GameStream (NVIDIA, 2015b) e Gaikai (GAIKAI, 2015), adquirida pela Sony para suportar seu serviço Playstation Now (MARTIN, 2015).

Como mostrado no item 2.4, diferentes tipos de aplicação 3D possuem diferentes tolerâncias à latência na perspectiva do usuário. O estilo de HPM para a qual se deseja apresentar uma solução de processamento remoto neste trabalho é caracterizado pela exploração e interação de ambientes, sem a necessidade de reações rápidas do usuário. Esse estilo de aplicação por si só apresenta maior tolerância à latência quando comparado com jogos de ação em primeira pessoa disponíveis no mercado

Com esse fator em mente, optou-se por adotar a arquitetura de processamento remoto com *streaming* de vídeo padrão dos serviços de *Cloud Gaming*, e agregar à solução o benefício da maior compatibilidade com ferramentas e serviços já disponíveis no mercado. Assim, o foco do tratamento dos efeitos da latência fica na interface com o usuário e na mecânica de interação da aplicação 3D dentro dos limites oferecidos pelos *Game Engines*.

Os componentes de *software* necessários para a implementação da solução podem ser categorizados por elemento da Arquitetura de Processamento Remoto para HPM, apresentada no item 2.2, da seguinte forma:

- **HPM –** No Servidor Remoto de Processamento da HPM: Trata-se de uma aplicação 3D que atende aos requisitos de HPM, apresentados no item 2.1, e é implementado com uma *Game Engine*. Esse *software* também é capaz de se comunicar com a Aplicação Cliente por intermédio do *Software* Gerenciador de Comunicação.
- **Codificador de *Streaming* de Vídeo –** No Servidor Remoto de Processamento da HPM: *Software* encarregado de fazer a captura das imagens geradas pela HPM, codifica-las em formato de *streaming* de vídeo e enviá-lo para a Aplicação Cliente.
- ***Software* Gerenciador de Comunicação** - No Servidor Remoto de Processamento da HPM: É o *broker* responsável por gerenciar mensagens trocadas entre a HPM e a Aplicação Cliente. Não precisa necessariamente estar no mesmo servidor da HPM.
- **Aplicação Cliente -** Dispositivo Cliente**:** Trata-se de um *software* para dispositivos móveis, compatível com a plataforma escolhida para os

testes, capaz de se comunicar com a HPM por intermédio do *Software* Gerenciador de Comunicação, receber e decodificar *streaming* de vídeo e receber interações do usuário.

Respeitando-se a mesma categorização, os componentes de *software* e *hardware* utilizados na construção da prova de conceito da solução foram:

- **Servidor Remoto de Processamento da HPM:** Computador *desktop* com sistema operacional Windows 10, HPM baseada no *Game Engine* Unreal Engine 4.10 e ferramenta de captura e codificação de vídeos Nvidia GameStream. As principais especificações de *hardware* do computador são: Intel Core i7 3770 3.4GHz, 16 gigabytes de memória RAM e placa de vídeo Nvidia GeForce GTX 980Ti com 6 gigabytes de memória VRAM.
- **Gerenciador de Comunicação:** *broker* RabbitMQ hospedado no mesmo computador do Servidor Remoto de Processamento da HPM.
- **Dispositivo Cliente:** *Tablet* Samsung Galaxy Tab S de 10,5 polegadas, modelo SM-T800 com sistema operacional Android 5.0.1.

A Figura 4 ilustra como esses componentes estão organizados na arquitetura da solução e como se comunicam entre si. Os retângulos em cinza-escuro representam os componentes de software: HPM, Codificador de Vídeo, Gerenciador de Comunicação e Aplicação Cliente. Os retângulos brancos representam módulos dos artefatos de *software* responsáveis por tarefas específicas como decodificação de vídeo e comunicação com o Gerenciador de Comunicação. As setas pretas indicam o fluxo de mensagens entre os componentes e os rótulos com contorno pontilhado mostram o tipo de mensagem. Os dois grandes retângulos preenchidos com hachura separam os componentes entre Cliente e Servidor.

A arquitetura trabalha de forma a proporcionar ao usuário uma experiência semelhante à de uma HPM sendo processada localmente em seu Dispositivo Cliente. A HPM, ou aplicação 3D, é processada no servidor e gera um conjunto de imagens. O Codificador de Vídeo faz a captura dessas imagens e gera um *streaming* de vídeo. Esse *streaming* é enviado de forma constante para o Dispositivo Cliente, mesmo que as imagens geradas pela HPM não apresentem variações entre si em algum momento. O módulo Decodificador de Vídeo da Aplicação Cliente recebe e decodifica

o *streaming* de vídeo recebido e mostra na tela do Dispositivo Cliente. O usuário vê na tela de seu dispositivo o conteúdo sendo gerado pela HPM no servidor como se o mesmo estivesse sendo processado localmente em seu dispositivo. Caso deseje interagir com a aplicação, o usuário realizará comandos que serão interpretados pela Aplicação Cliente e enviados para o Gerenciador de Comunicação através do Módulo de Comunicação. O Gerenciador de Comunicação encaminhará essa mensagem para o Módulo de Comunicação da HPM no servidor que, por sua vez, repassará os comandos para o ambiente 3D. O usuário verá as consequências de suas interações na tela de seu dispositivo assim que as novas imagens geradas pela HPM forem recebidas pela Aplicação Cliente.

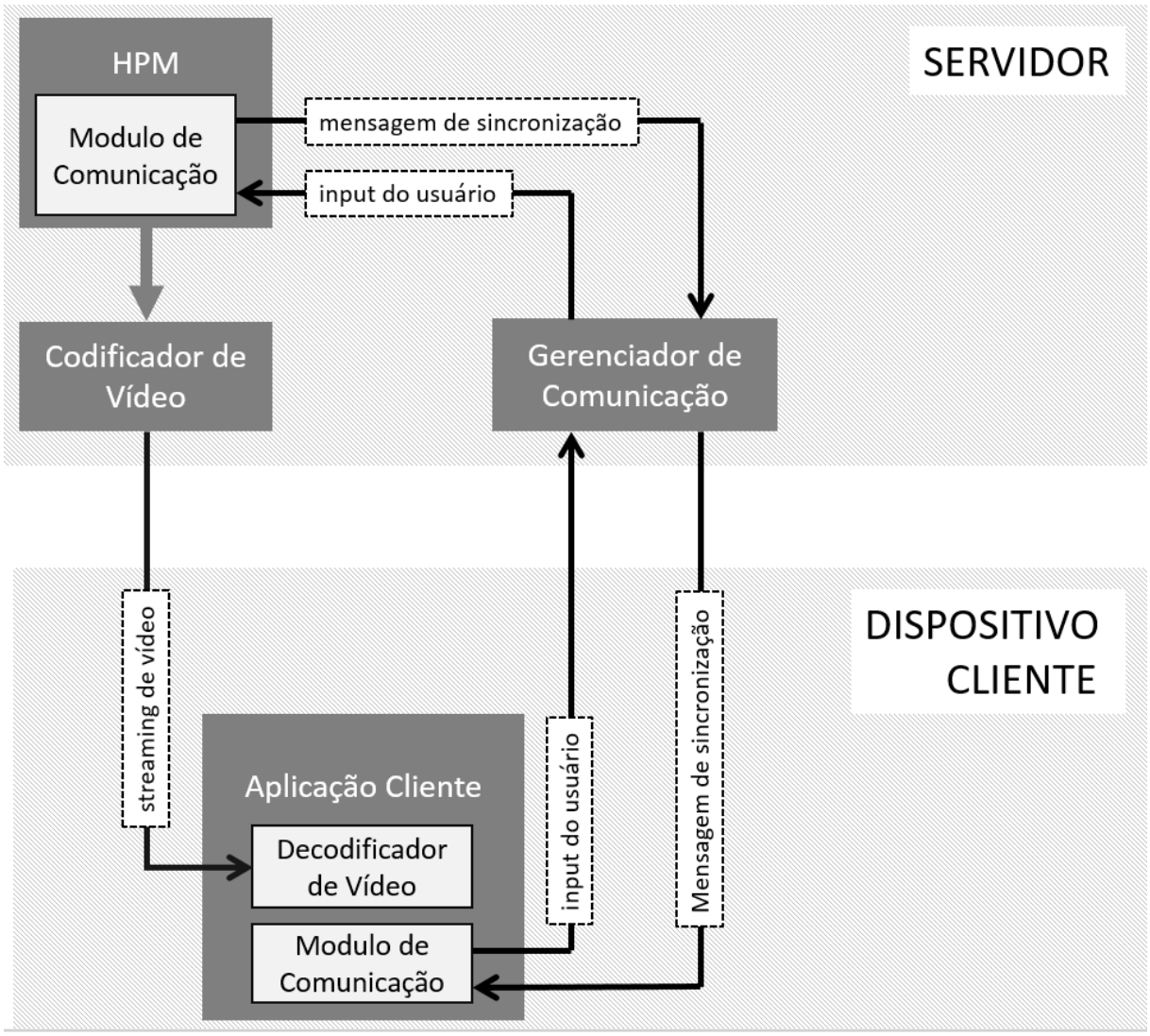

Figura 4 - Organização da Arquitetura de Processamento Remoto da Solução Desenvolvida (Fonte: próprio autor)

Além do *streaming* de vídeo, outros tipos de mídia podem ser mostrados na tela do Dispositivo Cliente em conjunto com a HPM. Nesse caso, são necessárias mensagens de sincronização geradas pela HPM no servidor para que a Aplicação Cliente saiba o momento exato de mostrar tais mídias.

O processo de determinação desses componentes de *software* e *hardware,* bem como detalhes de desenvolvimento, são descritos nas sessões a seguir.

4.2 Solução Alternativa e Solução Tradicional

Neste trabalho foi desenvolvida uma Solução de Processamento Remoto de HPM com recursos para redução dos efeitos da latência. Ao longo do desenvolvimento do trabalho, optou-se por utilizar meios específicos de interação entre o usuário e o dispositivo cliente para se atingir esse objetivo. Para possibilitar a avaliação dos resultados dessa solução, desenvolveu-se uma segunda versão que não emprega os meios de interação propostos na primeira, mas sim uma interface de usuário comumente usada em aplicações semelhantes à HPM desenvolvida. Detalhes sobre o desenvolvimento de ambas as soluções de interface são abordados no item 4.2. No contexto deste trabalho, a versão da solução que emprega os conceitos para redução de latência é referenciada como Solução Alternativa. A versão com interface comum é denominada Solução Tradicional.

Ambas utilizam a mesma arquitetura de processamento remoto descrita no item 4.1. No aspecto da implementação, as diferenças estão na forma como se comunicam com o servidor e como interagem com a lógica das interações da HPM, o que será abordado nos itens 4.4.3, 4.4.4 e 4.5.4.

4.3 Elaboração das Formas de Interação

Com a decisão de se usar padrões e ferramentas específicas de algumas soluções existentes de *Cloud Gaming* para o *streaming* de vídeo e implementação da HPM, como descrito no item 4.1, a Aplicação Cliente torna-se o elemento com maior

flexibilidade para a exploração de técnicas que possam reduzir os efeitos negativos da latência na experiência do usuário durante a interação.

Como abordado no item 4.4.3, foi definido um conjunto de interações possíveis na HPM implementada neste trabalho. Para que se pudesse explorar meios de se realizar essas interações que fossem preparados para lidar com efeitos negativos da latência, foram estabelecidas formas de interações básicas para cada uma delas com base no que é comumente adotado em aplicações semelhantes. Essas formas interações básicas serviram de base de comparação para o desenvolvimento das formas de interações específicas para situações de latência foram implementadas na Solução Tradicional. São descritos a seguir os processos de análise e definição das formas de interações da Solução Tradicional para cada interação definida para a HPM. Todos os aplicativos utilizados como exemplo são feitos para a plataforma móvel com *touchscreen.*

4.3.1 Interações da Solução Tradicional

- **Rotação da câmera**: Essa interação está presente em praticamente todas as aplicações 3D que permitem que o usuário controle o ponto de vista de seu personagem na tela. Na maioria dos jogos para *touchscreen,* a forma de interação utilizada para a rotação da câmera é através de uma alavanca analógica virtual posicionada em um dos cantos inferiores da tela. Essa solução simula o funcionamento de *joysticks* para jogos, que possuem uma alavanca analógica real para controle de rotação da câmera, e representa uma boa solução por não exigir que usuários de videogames tradicionais se adaptem com novos meios de interação para utilizar uma plataforma móvel com *touchscreen.*

  Uma outra forma de interação bastante usada para a rotação de câmera, baseia-se no gesto de *swipe*[4]. O usuário mantém o dedo sobre a tela enquanto o arrasta para a direção que deseja girar a imagem. Se ocorre um arrasto da direita para a esquerda, por exemplo, a câmera realizará uma rotação para a

[4] No contexto de interações com telas *touchscreen, swipe* é o nome dado a um tipo de gesto no qual se desliza o dedo rapidamente sobre a tela para qualquer direção. Esse gesto é comumente usado para navegar entre páginas e de uma aplicação.

direita de forma que o cenário acompanhe o movimento do dedo do usuário. Essa forma de interação é empregada nas versões móveis de aplicações como o Google StreetView, vídeos panorâmicos do Youtube e aplicações, Home Design 3D e Planner 5D Interior Design. Recentemente, jogos com visão em primeira pessoa vêm adotando essa forma de controle como uma alternativa opcional ou definitiva da alavanca virtual. Os exemplos mais populares na plataforma Android são Deer Hunter 2014, com três milhões e meio de *downloads* e Sniper 3D Assassin, com aproximadamente dois milhões de *downloads.*

Uma terceira forma menos convencional de controle de rotação de câmera é com uso dos sensores de aceleração e giroscópio dos dispositivos. Esse método é sempre oferecido como uma opção alternativa que pode ser escolhida pelo usuário e geralmente aparece em jogos de corrida como Alsphalt 5 e Real Racing.

Com base nesses dados, adotou-se o controle de rotação da câmera via *swipe* como forma de interação para a Solução Tradicional.

- **Translação da câmera**: Da mesma forma que o controle de rotação, o controle de translação da câmera está disponível na maioria das aplicações 3D que permitem que o usuário se desloque pelo cenário. As formas mais utilizadas para esse tipo de interação são a alavanca analógica virtual, como no item anterior, e teclas direcionais virtuais de pressionamento contínuo. A alavanca analógica permite que se controle a direção e a velocidade de deslocamento, enquanto que as teclas direcionais permitem o deslocamento apenas para direções e velocidade pré-definidas enquanto o usuário as mantém pressionadas. Jogos famosos na plataforma Android como Dead Trigger 2, NOVA 3 e Blitz Brigade, Sinper 3D Assassin e Deer Hunter 2014, todos com mais de um milhão de *downloads*, são exemplos que utilizam alavanca ou teclas direcionais de pressionamento contínuo para translação de câmera.

  Com base nesses dados, optou-se pela versão de teclas direcionais como método de translação da Solução Tradicional.

- **Seleção de objetos**: A seleção de objetos é um tipo de interação que não é muito comum em jogos com visão em primeira pessoa, que geralmente limitam-se a *gameplay* de tiro ao alvo ou controle de veículos. Exemplos mais comuns de aplicações com visão em primeira pessoa que envolvem a seleção de

objetos são as arquitetônicas, como Home Design 3D e Planner 5D Interior Design. Nesses aplicativos, a seleção de objetos ocorre por meio de um toque simples sobre o mesmo. Em jogos de estratégia com visão ortogonal, a seleção de objetos é um fator chave no *gameplay*. Clash of Clans, com mais de 20 milhões de *downloads,* Boom Beach, Clash of Kings e SimCity são exemplos que também utilizam o toque simples para seleção de objetos.

Adotou-se o toque simples como forma de seleção da Solução Tradicional com base nos dados acima.

- **Edição, adição e remoção de objetos**: A edição de objetos é um tipo de interação que inclui diversas possibilidades. Nas aplicações arquitetônicas exemplificadas nos itens anteriores, as interações possíveis são de mudança de cores, troca, remoção, translação e rotação. As translações são realizadas através do gesto *swipe*, com o qual o usuário pode arrastar os objetos pela cena. A rotação ocorre com um gesto *swipe* específico no qual desenha-se um círculo de forma contínua na tela no sentido em que se deseja girar o objeto. As demais edições são feitas através de opções em menus. Nos jogos de estratégia, exemplificados no item anterior, a translação de objetos também ocorre geralmente através do *swipe.* A rotação, quando existente, é feita através de opções de menus com incrementos de 45 ou 90 graus. Outros tipos de edição específicos do *gameplay* de cada título são geralmente realizados através de opções de menus.

  Para a Solução Tradicional, adotou-se opções de menus para as interações de troca de cor, remoção e edição de objetos. Para translação e rotação, utiliza-se o gesto de *swipe*.

Figura 5 – Imagem da interface da Solução Tradicional no Contexto de Navegação.
(Fonte: Próprio autor)

A Figura 5 mostra uma captura de tela feita no *tablet* Dispositivo Cliente durante a execução da aplicação na versão da Solução Tradicional. O círculo pontilhado mostra as teclas virtuais utilizadas para controlar a Translação de Câmera. O menu lateral, que só abre caso o usuário realize um gesto de *swipe* a partir do canto esquerdo da tela, mostra o conteúdo referente ao Contexto de Navegação.

Figura 6 - Imagem da interface da Solução Tradicional no Contexto de Movimento.
(Fonte: Próprio autor)

A Figura 6 mostra a mesma aplicação no Contexto de Movimento. Objetos selecionados ficam com um contorno brilhante. Se o modo de movimentação for ativado, surgem setas de rotação para que o usuário possa girar o objeto conforme o método de interação para Rotação de Objetos definido nesta sessão.

4.3.2 Interações da Solução Alternativa

Estudos indicam que atrasos de no mínimo 100 milissegundos são o suficiente para reduzir a performance de usuários de *Cloud Gaming* em até 25%, independentemente do estilo de jogo. Em jogos processados localmente com *multiplayer online* os efeitos negativos na performance dos usuários começam com valores de latência mais altos. Acredita-se que essa diferença ocorra pelo fato do atraso de resposta existir para todas as interações do usuário de *Cloud Gaming* com a interface do jogo (CLAYPOOL; FINKEL, 2014).

Considerando-se as *guidelines* de bom *design* de interface *software* propostos por Nielsen em 1994, apresentados no item 2.5.1, a interface com atrasos de um jogo

em *Cloud Gaming* pode ser prejudicada por não conseguir garantir os seguintes princípios:

- **Visibilidade do estado do sistema:** O usuário deve ser informado sobre o estado do sistema. Uma interface que atrasa as respostas esperadas pelo usuário para suas ações não deixa claro o que o sistema está fazendo naquele momento. Se o usuário deve esperar pela resposta, a interface deve informa-lo de que um determinado tempo de espera é necessário.
- **Consistência e padrões:** O sistema deve ser consistente para que o usuário saiba o que esperar como resposta por suas ações. Uma interface com latência pode não ser capaz de cumprir esse princípio, uma vez que repostas que deveriam ser imediatas sofrem atrasos. Se a latência for variável, a previsão por parte do usuário torna-se ainda mais difícil.

Em um trabalho realizado em 2008 (BADASHIAN et al., 2008), é proposto e validado um conjunto de *guidelines* para *design* de interface de *software* que inclui os princípios básicos de Nielsen e acrescenta alguns novos. Dentre eles, é apresentado um que sugere especificamente que se tome cuidado com a latência, exibindo-se informações aos usuários que os permitam saber que o sistema não travou.

Com base nessas observações, buscou-se identificar em cada uma das formas de interação da Solução Tradicional definidas no item 4.3.1 características que desrespeitassem as *guidelines* destacadas acima em uma situação de latência. Em seguida, foram propostas alternativas que estivessem de acordo com elas. Esse processo é apresentado a seguir, categorizado por tipo de interação definido para a HPM do projeto:

- **Rotação da Câmera:** O gesto de *swipe,* utilizado na Solução Tradicional, permite que o usuário tenha controle preciso sobre a rotação da câmera. Ao colocar o dedo sobre um ponto da tela e deslizá-lo, a região da imagem sobre a qual o dedo foi colocado acompanhará seu movimento enquanto o usuário o mantiver na tela. Em uma situação de latência, essa região não acompanhará o dedo em tempo real e o usuário perde a precisão do *feedback* de seus gestos. Por consequência, a capacidade de se prever o comportamento do sistema fica prejudicada.

  Uma possível forma de se eliminar o problema da imprecisão do *feedback* em tempo real é a sua substituição por outra forma de resposta que

permita ao usuário prever o comportamento da câmera. Uma opção considerada foi o uso de teclas direcionais de toque simples, que promovem a rotação da câmera de forma incremental. Se os toques nessas teclas sempre causam incrementos de rotação de 10 graus na direção indicada por cada tecla, por exemplo, o usuário tem condições de prever o resultado de seus toques. O tempo de espera pela rotação após o toque em uma tecla deve ser indicado de alguma forma para que se cumpra a *guideline* de Visibilidade do Estado do Sistema. Um ponto negativo dessa solução é que a precisão permitida nas rotações diminui com o aumento do tamanho dos incrementos e diminuição da quantidade de teclas. Incrementos pequenos e grande quantidade de teclas, por outro lado, deixam o processo de rotação mais lento e complicado.

Em um trabalho feito sobre os fatores que afetam a performance e percepção de jogadores em jogos *multiplayer* é notado que o efeito da latência é menor na maioria dos jogos de estratégia porque eles costumam adotar uma forma de controle indireta (DICK; WELLNITZ; WOLF, 2005). Ao invés de controlar o processo de movimentação dos personagens, o jogador apenas indica um destino e o jogo se responsabiliza por leva-lo até lá. Abordagem semelhante pode ser aplicada em uma possível solução de rotação com toques simples. Ao invés de teclas direcionais, o usuário poderia tocar sobre a região da tela na qual ele deseja que a câmera se centralize e a aplicação se responsabilizará por girar a câmera de forma adequada, provocando uma rotação automática.

O *swipe* e a alavanca analógica virtual ainda foram considerados como possíveis soluções para a Interação de Rotação, mas após análise dos prós e contras de cada opção, definiu-se o que a rotação automática seria a melhor opção. A Figura 7 mostra o gráfico *Design Rationale* dessa decisão.

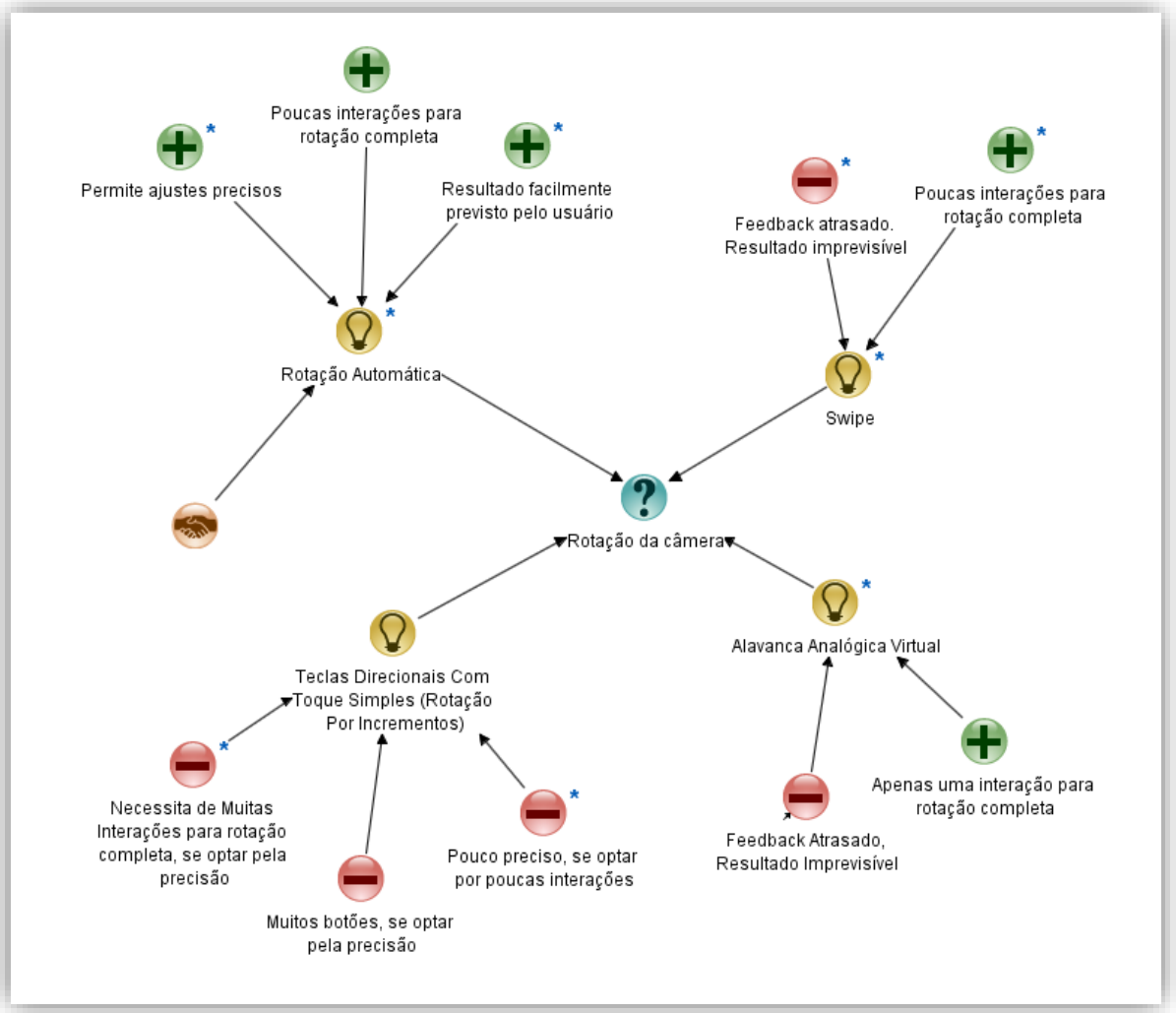

Figura 7 – Gráfico *Design Rationale* da decisão de interação de Rotação de Câmera da Solução Alternativa (Fonte: Próprio autor)

- **Translação da Câmera:** As teclas direcionais de pressionamento contínuo, utilizadas na Solução Tradicional, possuem problemas semelhantes ao *swipe* e alavanca analógica virtual na rotação de câmera. As três são formas de interação contínua que dependem do *feedback* instantâneo da aplicação para que o usuário tenha controle preciso. No caso das teclas direcionais, a latência causará atraso no início e no fim da movimentação do personagem nos momentos em que o usuário inicia e termina o pressionamento das teclas, respectivamente. Como esse método de interação não dá controle sobre a velocidade nem sobre direções de movimento além das oferecidas por cada tecla, o usuário precisa realizar combinações de pressionamentos para controlar a câmera até o destino desejado. Sem *feedback* instantâneo, a

capacidade de previsão da movimentação da câmera em cada interação fica prejudicada.

Assim como na rotação, a substituição do controle contínuo por um tipo de controle discreto, com toques simples e incrementais, pode minimizar o problema. Uma possível solução considerada foi o uso das mesmas teclas direcionais, mas com a troca do pressionamento contínuo por toques simples. Nesse método, a cada toque em uma tecla, a câmera se move para a direção correspondente por uma distância padrão fixa. Dessa forma, o usuário consegue prever a reação da câmera a cada toque, mesmo que tenha que esperar um período de tempo para que essa reação ocorra. Um ponto negativo dessa opção é que a movimentação por distâncias grandes exige uma grande quantidade de interações, resultando em vários ciclos de toque e espera por parte do usuário.

O conceito de controle autônomo utilizado na rotação de câmera automática também pode ser aplicado na translação. Nesse caso, o usuário toca sobre uma região do ambiente para o qual ele deseja transladar a câmera e a aplicação realiza o conjunto de translações necessárias para que o destino seja alcançado. Essa opção possibilita que as translações para pontos específicos do cenário sejam concluídas com poucas interações. Porém, a movimentação para regiões fora do campo de visão do jogador necessitam interações de rotação prévias, para que se visualize o destino desejado sobre o qual o toque deve ocorrer.

As opções de alavanca analógica e teclas direcionais com pressionamento contínuo também foram considerados como possíveis soluções de Translação de Câmera. Após análise das vantagens e desvantagens de cada opção, definiu-se a translação de câmera automática como melhor solução. A Figura 8 mostra o gráfico *Design Rationale* dessa decisão.

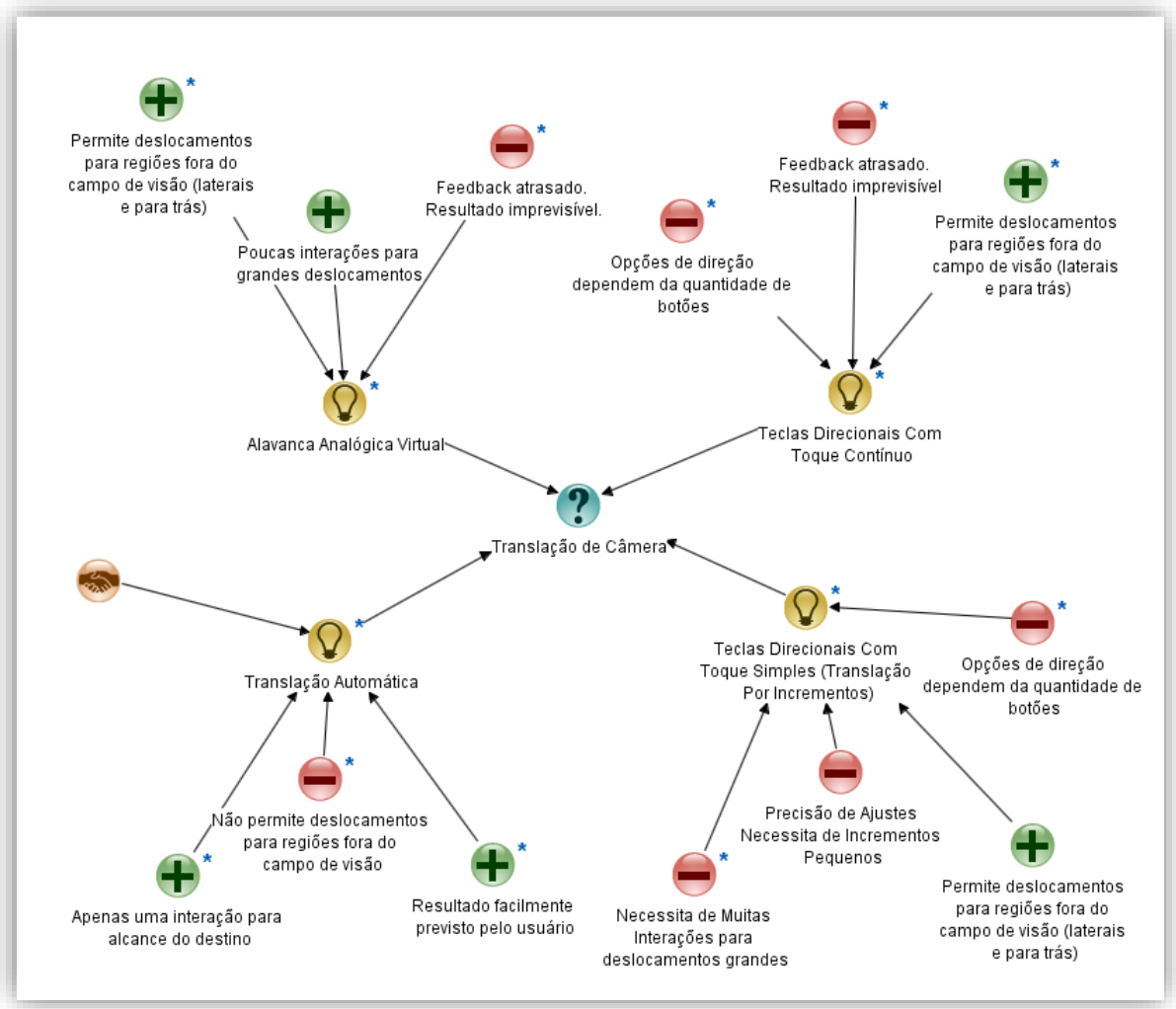

Figura 8 - Gráfico *Design Rationale* da decisão de interação de Translação de Câmera da Solução Alternativa (Fonte: Próprio autor)

- **Seleção de objetos:** O toque simples utilizado na Solução Tradicional para a seleção de objetos já possui a característica discreta das interações definidas para rotação e translação na Solução Alternativa. Assim, o único problema desse método em situações de latência é o atraso da resposta do toque do usuário, que deve ser devidamente informado para que se cumpra a *guideline* de Visibilidade do Estado do Sistema.

  Foi considerada também uma solução para seleção de objetos baseada em lista, na qual o usuário tem acesso a uma lista com todos os objetos selecionáveis na cena e não interage diretamente com o ambiente para selecioná-los. Esse método também é baseado em toques simples, mas

dificulta a identificação de objetos específicos no campo de visão do usuário, principalmente em situações em que existem várias cópias do mesmo objeto na cena.

O toque simples sobre os objetos foi mantido como método de interação definitivo para a seleção de objetos na Solução Alternativa. A Figura 9 mostra o gráfico *Design Rationale* dessa decisão.

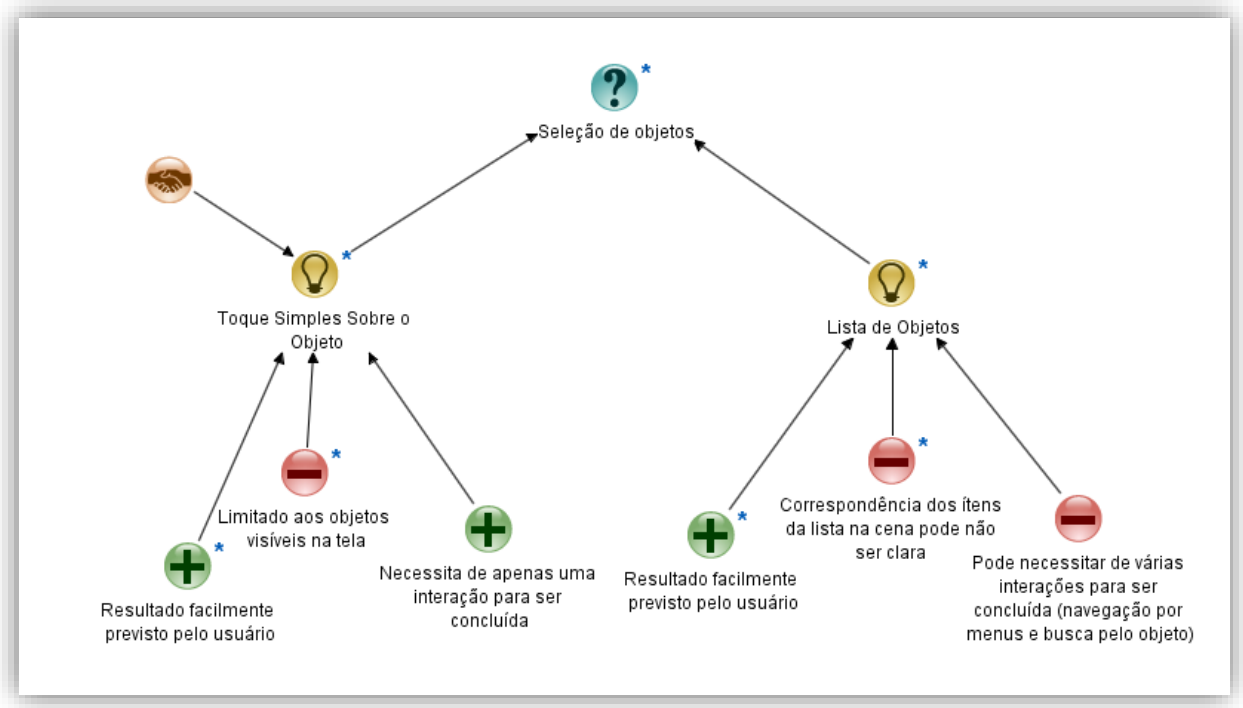

Figura 9 - Gráfico *Design Rationale* da decisão de interação de Seleção de Objetos da Solução Alternativa (Fonte: Próprio autor)

As três formas de interação definidas acima utilizam o toque simples do usuário como entrada. **Rotação da Câmera, Translação da Câmera**, e **Seleção de Objetos** podem ocorrer no mesmo contexto do *gameplay*, o que impossibilita o compartilhamento do mesmo gesto entre eles. Atendo-se à característica discreta do toque simples, definiu-se mais dois gestos que mantivessem esse aspecto. A Rotação de Câmera é realizada com um único **Toque Simples**. A Translação da Câmera é feita com um **Toque Duplo** e a Seleção de Objetos é ativada pelo **Toque Longo.**

Para garantir a *guideline* de Visibilidade do Estado do Sistema, utilizou-se a vibração do dispositivo como forma de *feedback*. Após cada um desses gestos, o

dispositivo vibra de forma suave por um período correspondente ao tempo de resposta do servidor a mensagens enviadas em interações anteriores. O momento em que a vibração termina deve coincidir aproximadamente com o momento em que a os resultados da interação do usuário com aplicação são mostrados no vídeo.

O cancelamento da seleção de objetos é feito através do toque longo sobre qualquer região da tela ou através do botão Voltar no menu do contexto de edição. Objetos selecionados ficam com um contorno brilhante para indicar ao usuário que estão em modo de edição.

- **Edição, adição e remoção de objetos:** Das ações possíveis nesse conjunto, adição, remoção e alteração de cores e tipo dos objetos são feitas através de menus. Optou-se por utilizar o mesmo estilo de menu da Solução Alternativa na Solução Tradicional, que consiste na gaveta lateral padrão do Android e adotada em diversos aplicativos (GOOGLE, 2015). A gaveta lateral possibilita interações através de dois tipos de gesto: Toque simples, para seleção de itens, e *swipe*, para abertura e fechamento da gaveta e rolagem de listas. No caso dos toques simples, pode ser adotado o mesmo estilo de *feedback* vibratório dos itens anteriores. O gesto de *swipe* exige outra alternativa pelos mesmos motivos identificados nas interações de rotação e translação de câmera.

  Uma segunda alternativa, inspirada em soluções utilizadas em trabalhos anteriores, é o processamento dos menus diretamente no Dispositivo Cliente. Dessa forma, o usuário pode ter retorno visual instantâneo de suas interações com o menu. A rolagem de uma lista de opções, por exemplo, pode ocorrer através do gesto de *swipe* normalmente, sem que haja atraso no *feedback* visual dessa interação. O mesmo vale para a abertura e fechamento do menu, através do *swipe* lateral, e pressionamento de botões. Porém, essas vantagens valem apenas para as interações com o menu em si. O tempo para uma escolha feita através do menu ter reflexo no ambiente 3D continua dependendo da latência da solução. Para garantir a Visibilidade do Estado do Sistema, animações no próprio menu podem servir de indicativo visual do tempo de espera. Um botão de inserção de um objeto, por exemplo, pode ter uma animação de pressionamento que dure tanto quanto a latência de resposta do servidor para ser concluída. Outra situação na qual pode ser utilizada uma

animação com essa função é durante a troca de menus quando ocorrem mudanças de contexto de *gameplay*, como explicado no item 4.4.3.

Um fator importante a ser levado em conta na decisão entre as duas opções é o impacto de cada uma na modularidade da solução. Os menus processados remotamente permitem que a Aplicação Cliente trate apenas de diferentes tipos de entrada de usuário e envie os comandos correspondentes para o servidor. Dessa forma, a Aplicação Cliente pode ser utilizada para comunicação com diferentes HPMs que utilizem os mesmos tipos de interações. Com os menus processados localmente, cria-se uma dependência entre as duas aplicações, uma vez que o conteúdo e organização dos menus dependem da HPM. Com exceção das que utilizam um mesmo padrão de menus, aplicações 3D diferentes não serão compatíveis com a mesma Aplicação Cliente da Solução Alternativa. Uma possível solução para essa limitação é a implementação de um módulo que tenha condições de modificar conteúdo e forma dos menus da Aplicação Cliente de acordo com a HPM com a qual ela se comunica.

Neste trabalho, os menus da aplicação são compostos apenas por botões com textos e imagens. Contudo, a solução de menus processados localmente pode conter diversos tipos de mídia que permitem interações livre de latência. Uma lista de móveis, por exemplo, pode conter animações que mostrem cada objeto de diferentes ângulos ao invés de imagens fixas. Em uma aplicação de exploração de um museu, um objeto do ambiente 3D remoto poderia ter uma versão 3D simplificada mostrada no menu local, de forma que o usuário pudesse manipulá-lo e ter acesso a textos e áudio descritivos sem latência, por exemplo.

A escolha da solução de menus processados localmente é representada no gráfico *Design Rationale* da Figura 10.

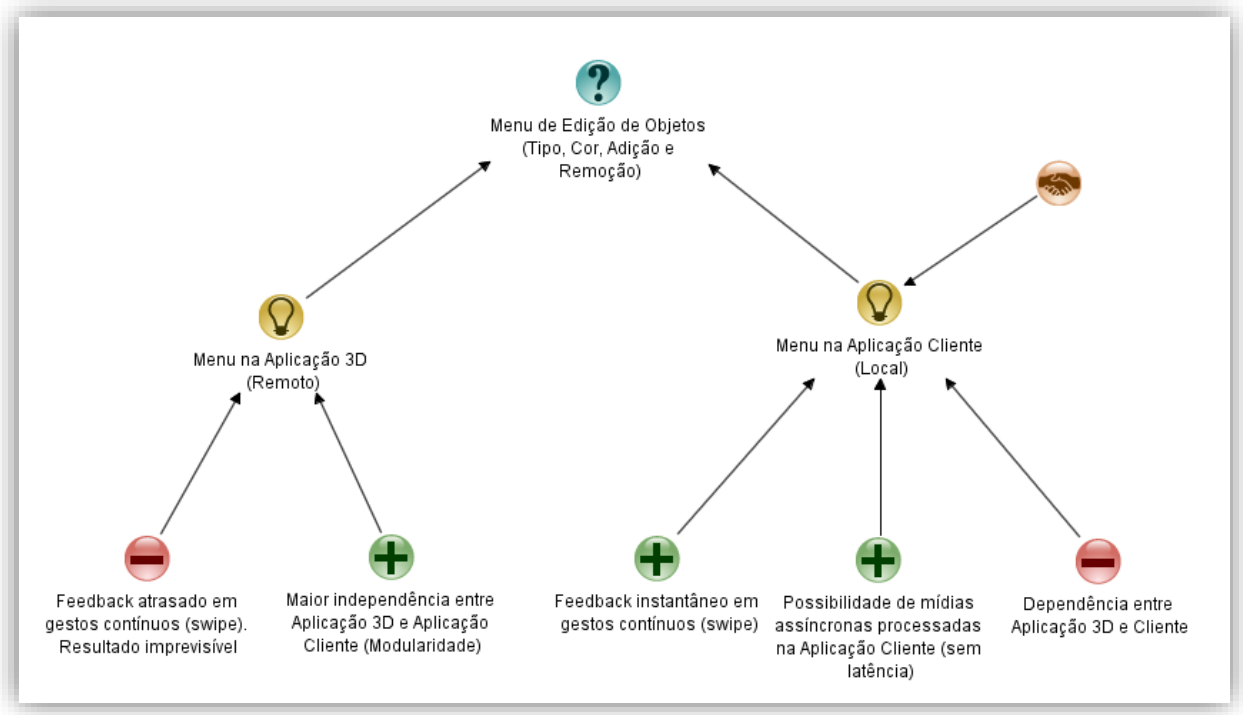

Figura 10 - Gráfico *Design Rationale* da decisão de menus da Edição de Objetos da Solução Alternativa (Fonte: Próprio autor)

Na Solução Tradicional, as ações de translação de objetos são realizadas através de gestos *swipe*. Usou-se a mesma abordagem da Rotação de Câmera para determinação da forma de interação da Solução Alternativa. Para movimentação do objeto o *swipe* foi substituído pelo mesmo gesto utilizado na seleção de objetos: Toque Longo. O usuário toca sobre a região do cenário na qual pretende posicionar o objeto selecionado, e a aplicação se encarrega de fazer a translação automática do mesmo para o destino desejado. O compartilhamento de gestos entre diferentes ações é um dos motivos da utilização de contextos de *gameplay* na Solução Alternativa. Se o usuário escolhe a opção “mover objeto” no Menu de Edição de objetos, o contexto do *gameplay* muda de Contexto de Edição para Contexto de Movimento e o Toque Longo deixa de ter a função de selecionar objetos. Para voltar ao Contexto de Edição, o usuário deve selecionar a opção Cancelar Movimentação no Menu de Movimentação.

A rotação de objetos também é realizada através de gestos *swipe* na Solução Tradicional. Porém, com gestos específicos a partir de ícones de rotação que aparecem ao redor do objeto selecionado. Para a Solução Alternativa, optou-se novamente por uma solução análoga à Rotação de Câmera, mas com toques longos feitos sobre os mesmos ícones de rotação utilizados na Solução Tradicional. Se o

usuário deseja girar o objeto no sentido horário, realiza um toque longo sobre o ícone que aponta para esse sentido. A aplicação então realiza um giro incremental de 45 graus no objeto. A rotação só fica disponível no Contexto de Movimento do *gameplay.*

Durante o processo de implementação dessas interações, notou-se em testes preliminares que a incapacidade de deslocamento para regiões fora do campo de visão na forma de Translação de Câmera automática poderia prejudicar a experiência de navegação. Como forma de se contornar essa limitação, criou-se um conjunto adicional de interações que oferecessem essa possibilidade.

- **Recursos adicionais para Translação e Rotação de Câmera:** Foram adicionadas margens transparentes ao redor de toda a tela com cerca de um centímetro de largura. São realizadas translações incrementais da câmera para diferentes direções caso o usuário faça um toque longo sobre essas regiões. A região de baixo da tela permite um deslocamento incremental de cerca de dois metros para trás. A região de cima causa um deslocamento de um metro para a frente. As regiões laterais possibilitam deslocamentos laterais com incrementos de um metro. Além dessas regiões, foi adicionado o reconhecimento de um gesto rápido de *swipe* em qualquer local da tela para ativação de um giro incremental de 180 graus na câmera. Essa interação permite uma inversão rápida de direção da câmera sem a necessidade de diversos toques duplos na tela.

  Apesar da possível agilidade adicionada por esses recursos adicionais, o aumento de interações possíveis aumenta a complexidade da interface como um todo, o que pode torna-la ainda menos intuitiva do que uma interface tradicional, aumentando a curva de aprendizado.

Figura 11 –Imagem da interface da Solução Alternativa no Contexto de Movimento.
(Fonte: Próprio autor)

A Figura 11 mostra uma captura de tela feita no *tablet* Dispositivo Cliente durante a execução da aplicação na versão da Solução Alternativa. As margens transparentes são utilizadas para movimentos de translação incrementais. A rotação e a translação automáticas ocorrem apenas por meio de gestos.

4.3.3 Fluxograma de Contextos de *Gameplay*

A mecânica de interação com a HPM deste projeto é organizada através de quatro contextos de *gameplay* como foi abordado parcialmente nos itens 4.4.3 e 4.5.2: *ContextNavigation,* ou Contexto de Navegação; *ContextInsertion*, ou Contexto de Inserção; *ContextEdition*, ou Contexto de Edição; e *ContextMove,* ou Contexto de Movimento. Nos códigos da HPM e da Aplicação Cliente são adotadas as nomenclaturas em inglês.

O *gameplay* da aplicação pode estar em qualquer um desses contextos, mas nunca em mais de um ao mesmo tempo. A troca de contextos é causada por

determinadas ações do usuário e causa mudanças no conteúdo do menu, que se adapta ao contexto atual, e nas possibilidades de interação com o ambiente 3D.

A Figura 12 mostra todas as ações que causam mudança de contexto em forma de fluxograma.

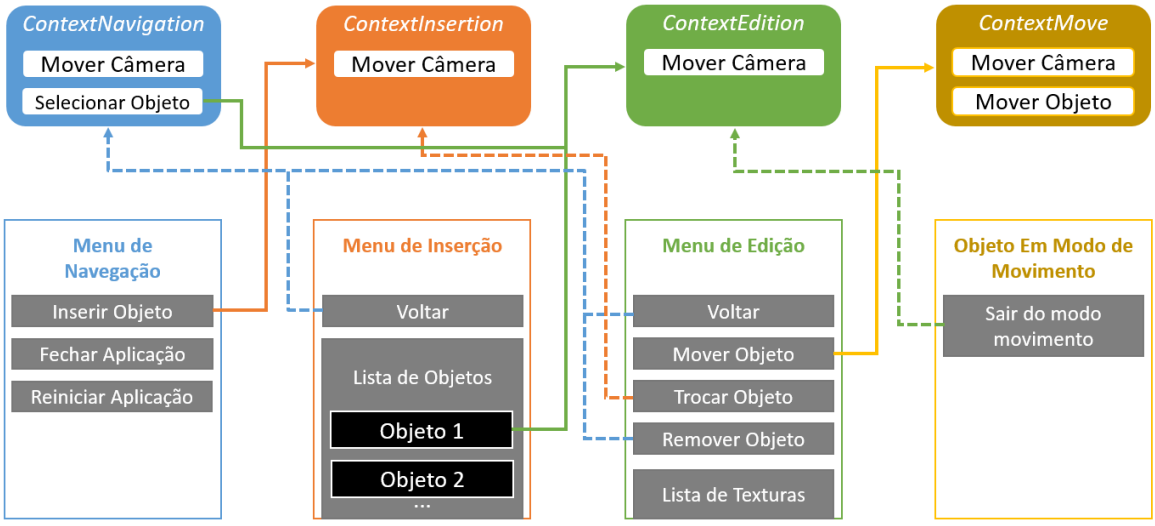

Figura 12 - Fluxograma de Contextos do *Gameplay* da HPM (Fonte: Próprio autor)

Os quatro retângulos com cantos arredondados representam os contextos e dentro deles estão as interações com o ambiente 3D permitidas em cada um. Os retângulos logo abaixo, com cantos retos, representam o conteúdo mostrado pelo menu em cada contexto.

Tanto interações com o ambiente 3D quanto botões de menu são capazes de alterar o contexto da aplicação. Essas relações são representadas pelos caminhos contínuos e pontilhados, sendo que os pontilhados indicam caminhos no sentido da direita para a esquerda e os contínuos da esquerda para a direita.

Essa mecânica é a mesma para as duas soluções de interface. A diferença está apenas na implementação. Na Solução Tradicional a lógica interna dos menus fica completamente a cargo da HPM, como explicado no item 4.4.3. Na Solução Alternativa a lógica interna dos menus fica na Aplicação Cliente, como explicado no item 4.5.2.

Capítulo 5

4.4 Construção da HPM

A HPM construída e utilizada na solução proposta neste trabalho consiste em um ambiente arquitetônico interativo que atende aos requisitos que definem HPM apresentados no item 2.1. Considera-se que essa aplicação específica é um bom representante de um estilo de aplicação 3D de exploração, sem reações rápidas do usuário, que pode ser utilizada na simulação de museus, showrooms, laboratórios, etc. Dessa forma, as soluções de interatividade propostas neste trabalho podem ser aproveitadas em aplicações que compartilham dessas características, mas talvez não sejam ideais para aplicações mais frenéticas como jogos de tiro em primeira pessoa, por exemplo.

A implementação dessa HPM dependeu de uma série de etapas descritas nos itens a seguir.

4.4.1 Escolha do *Game Engine* e Ferramenta de Codificação

A viabilização da HPM planejada dependeu da escolha de um *Game Engine* capaz de proporcionar as características que definem essa mídia, ou seja, que tire proveito de computadores *desktop* de última geração para gerar gráficos realistas e permita interatividade. Qualquer *Game Engine* é capaz de prover interatividade, mas para garantir o requisito de alta qualidade gráfica buscou-se pelas opções utilizadas nos jogos mais modernos e graficamente avançados no momento em que essa etapa foi realizada. Outro fator levado em conta na seleção foi o preço. Há muitos *Game*

Engines utilizados em jogos topo de linha que são de propriedade da própria desenvolvedora e possuem alto custo de licenciamento. Mas há também opções gratuitas ou com modelos de licenciamento viáveis no escopo deste trabalho. Dentre essas, as mais famosas são: Unreal Engine 3 (EPIC GAMES, 2013), Unreal Engine 4 (EPIC GAMES, 2014), CryEngine 3 (CRYTEK, 2014) e Unity 5 (UNITY, 2015).

Considerando-se as características da HPM utilizada nesse trabalho, O Unreal Engine 4 possui uma potencial vantagem em relação as outras opções. Desde seu lançamento, em 2013, esse motor gráfico tem atraído interesse da área de Visualizações Arquitetônicas (PLANTE, 2015). Um dos motivos são os recursos de cálculo de iluminação do *engine*, que se assemelham bastante aos utilizados por *software* de pré-processamento, possibilitando a geração de imagens com qualidade muito semelhante ao que se busca nessa área (SAVAGE, 2014). Outro motivo pode estar relacionado à facilidade de uso do *engine* por pessoas que não dominam linguagens de programação e o fato de sua licença ser gratuita para fins de visualizações de ambientes. *O* interesse foi tanto, que a própria Epic Games, desenvolvedora da Unreal Engine, e a Nvidia, líder no mercado de GPUs (FORBES, 2015a) promoveram um concurso para artistas da área (BEKERMAN, 2015) (EPIC GAMES, 2015a) .

Além desses motivos, esse motor gráfico assemelha-se à sua antiga versão, Unreal Engine 3, utilizada e dominada pelo autor em trabalho anterior para desenvolvimento de técnicas de conversão de ambientes pré-processados em HPM (RABELLO, 2013), tornando o Unreal Engine 4 a escolha para a solução proposta neste trabalho.

Este motor gráfico possui suporte para Windows, Linux e Mac OS X. Sendo que a API DirectX é suportada apenas no Windows. Como a ferramenta de codificação selecionada para o projeto depende dessa API, Windows foi a plataforma escolhida para o Servidor da arquitetura deste projeto.

Para captura e codificação de vídeo, optou-se pelo *framework* da Nvidia, utilizado pela Amazon em sua solução *AppStream* e *Cloud Gaming* (AMAZON, 2015b), pela Otoy em sua solução X.IO (OTOY, 2015) e pela própria Nvidia no serviço Nvidia GameStream (NVIDIA, 2015b). Esse *framework* permite a captura de imagens diretamente do *Frame Buffer* do GPU (*Graphics Processor Unit)* e utiliza instruções de *hardware* da própria placa gráfica para codificar a sequência de imagens em *streaming* de vídeo no formato h264. O recurso é disponibilizado pela Nvidia tanto em

suas placas de vídeo profissionais para servidores, quanto para jogos em computadores *desktop*. Na versão de *desktop,* a funcionalidade é chamada de GameStream e está presente nos *drivers* oficiais das placas de vídeo Nvidia para Windows. Há também uma aplicação oficial para decodificação do *streaming* de vídeo em dispositivos Android da linha Nvidia Shield (NVIDIA, 2015c) e uma biblioteca *open source* não oficial para plataformas Android, Windows, Mac, Linux, iOS, Raspberry Pi e SamsungGear VR (MOONLIGHT, 2015). Neste projeto, utilizou-se um computador *desktop* com uma placa de vídeo Nvidia GTX 980Ti como servidor e a funcionalidade de GameStream do driver de vídeo como ferramenta de captura e codificação. O *framework* possui APIs de comunicação próprias para envio do *streaming* de vídeo e recebimento de entradas do usuário no cliente.

A construção da HPM pode ser dividida em três etapas: Modelagem do Ambiente 3D, Implementação da Lógica de Interação e Implementação das Funcionalidades de Comunicação.

4.4.2 Modelagem do Ambiente 3D da HPM

O Unreal Engine 4 possui um editor que permite a organização do conteúdo 3D e criação da lógica através de uma ferramenta gráfica denominada *Blueprint Editor*. Qualquer funcionalidade que não esteja disponível por padrão nessa ferramenta, pode ser implementada na linguagem C++ com a ferramenta Microsoft Visual Studio, que possui integração oficial com o motor gráfico. Apesar da capacidade de manipulação de conteúdo 3D do Unreal dar liberdade para que se construa cenários com diversos recursos, ela não é feita para a produção de modelos 3D, mas sim para a organização de modelos importados de outras ferramentas específicas. Para a produção do conteúdo 3D da HPM deste trabalho, optou-se pelo *software* Autodesk 3D Studio Max. A escolha dessa ferramenta não interfere no *workflow* do *Game Engine*.

Foi adquirido um ambiente arquitetônico 3D - Scene 02 AIC01 - para o *software* Autodesk 3D Studio Max no site da Evermotion (EVERMOTION, 2014). Esse modelo é feito para renderização com o *plugin* de 3D Studio para pré-processamento chamado VRay. Foi feita a remodelagem completa do ambiente para torna-lo compatível com a abordagem de processamento em tempo real e exportação para o Unreal Engine 4. Durante a remodelagem, foram utilizadas e aperfeiçoadas técnicas de remodelagem de cenas para pré-processamento em cenas para processamento

em tempo real propostas pelo autor (RABELLO, 2013), dentre as quais as principais são descritas abaixo:

- **Otimização de Vegetação:** Originalmente desenvolvida para reduzir a quantidade de polígonos em modelos 3D de vegetação feitos para pré-processamento sem causar redução visual na densidade de folhas. É utilizada uma técnica baseada em polígonos com opacidade, na qual são utilizados polígonos intercalados que recebem texturas com mapas de opacidade para mascarar sua posição. Dessa forma, permite-se que um conjunto de galhos e folhas seja representado por apenas uma face, como ilustrado na **Figura 13**. Neste trabalho, a técnica foi adaptada para redução da quantidade de polígonos em superfícies com pelos ou fios. A **Figura 14** ilustra a técnica adaptada para o modelo de carpete do ambiente remodelado. Em modelos para pré-processamento, a reprodução de pelos e fios é feita com geometria, com um cilindro de vários polígonos para cada fio. Esse tipo de modelagem gera grande quantidade de polígonos em uma mesma cena, o que pode ser custoso para o processamento em tempo real.

Figura 13 - Ilustração da Técnica de Polígonos com Opacidade. (Fonte: Próprio autor)

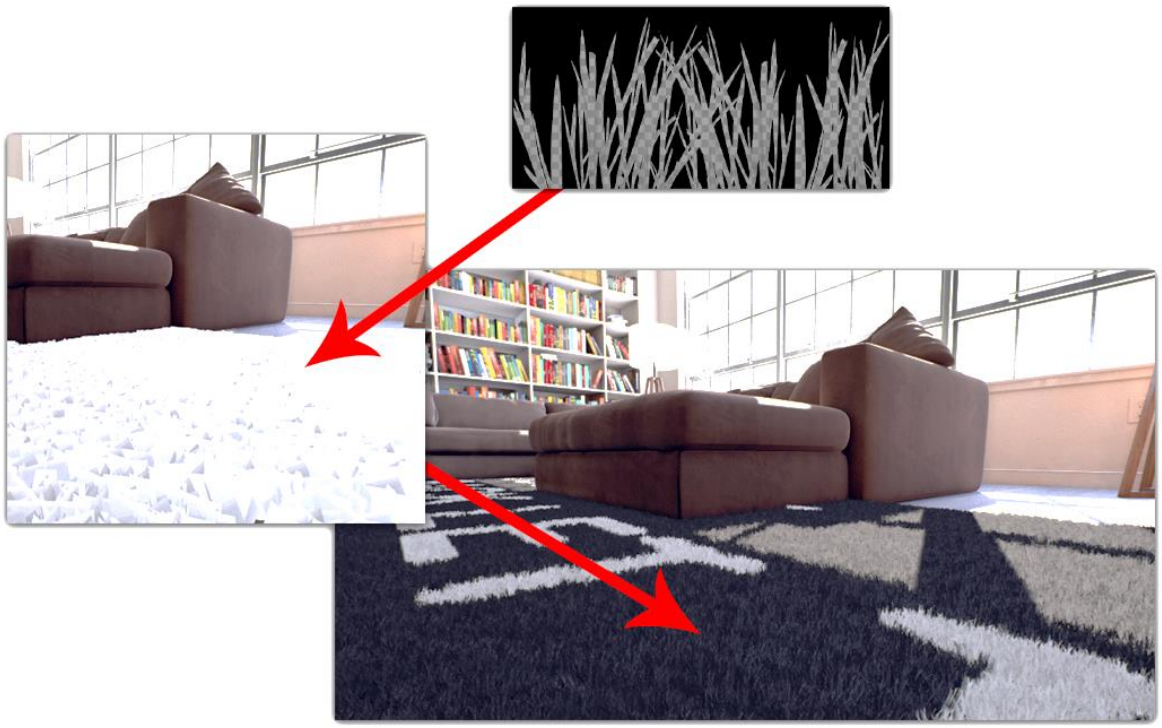

Figura 14 - Ilustração da Técnica de Polígonos com Opacidade aplicada para simulação de pelos no carpete. (Fonte: próprio autor)

- **Otimização de Área de Superfícies:** Nesta técnica, todos os polígonos que nunca serão visualizados pelo usuário durante a interação com o ambiente 3D são removidos. Já na abordagem de pré-processamento essa preocupação não existe, em tempo real a quantidade de memória utilizada é um fator crítico no desempenho da aplicação. O Unreal Engine 4 aplica texturas de iluminação sobre as texturas de cor de todos os modelos da cena, incluindo as partes que não estão visíveis. Como resultado, ambientes com muitos polígonos escondidos desperdiçarão memória de vídeo. A remodelagem de todos os objetos da cena incluiu essa técnica.
- **Mapeamento para Iluminação:** O mapeamento de um modelo 3D é o procedimento que determina a maneira como as texturas serão projetadas sobre sua superfície. Existem diversas técnicas de mapeamento utilizadas para compor o material da forma desejada em um modelo. O Unreal Engine 4 necessita de objetos com dois canais de mapeamento. O primeiro é utilizado para as texturas e é idêntico ao existente nos modelos para pré-

processamento. O segundo é utilizado pela ferramenta de cálculo de iluminação do motor gráfico, e possui o requisito de não ter faces sobrepostas. Esse mapeamento não é padrão na abordagem de pré-processamento do 3D Studio, e teve que ser adicionado em todos os objetos da cena da Evermotion durante a remodelagem. A Figura 15 compara o mapeamento comum de um modelo 3D com o mapeamento sem sobreposição de faces.

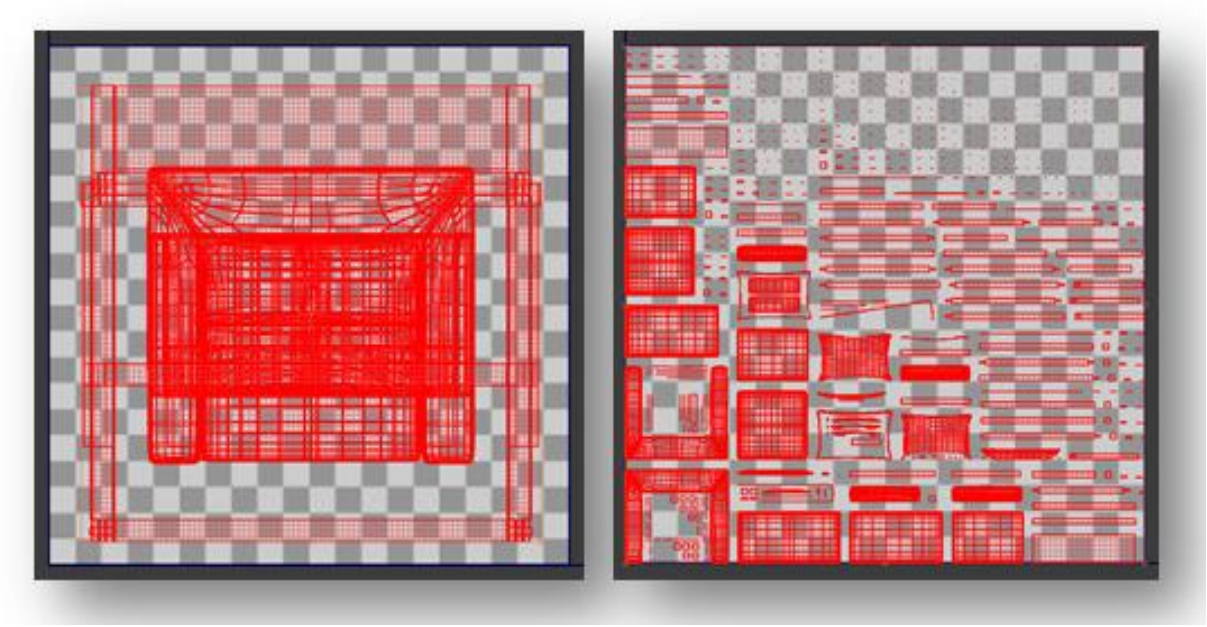

Figura 15 - Mapeamento com sobreposição à esquerda, com diversos polígonos de um mesmo objeto sobrepostas e ocupando a mesma área da textura quadriculada. Mapeamento sem sobreposição à direita, no qual cada polígono do objeto ocupa uma região distinta de uma mesma textura. (Fonte: Próprio autor)

- **Exportação para o Unreal:** O formato de arquivo reconhecido pelo Unreal Engine 4 na importação de objetos é o FBX (AUTODESK, 2015b). Além da malha 3D dos objetos que é triangulada automaticamente pelo plug-in FBX do 3D Studio, os seguintes itens também são reconhecidos na importação:
  - **Múltiplos canais de materiais[5] por objeto**: A cada face da malha 3D pode ser relacionado um identificador de material diferente. Os materiais são relacionados a esses identificadores no *Game Engine.*

[5] No contexto de modelagem 3D para processamento em tempo real ou pré-processamento, material, ou *shader,* são um conjunto de instruções que indica como a luz deve interagir com a superfície do objeto.

- **Sólidos de colisão:** Cada objeto do arquivo pode ter um único sólido de colisão que é identificado pelo Unreal Engine Editor através do seu nome que deve seguir o padrão “UCX_nomeDoObjeto”. Os sólidos de colisão são estruturas que devem envolver qualquer malha 3D que terá algum tipo de interação com o usuário ou outro sólido de colisão no momento do *gameplay*. Todos os objetos que devem ter interação com o usuário na cena remodelada, como móveis, paredes e piso, tiveram seus respectivos sólidos de colisão modelados.

- **Texturização no Unreal Engine:** Os materiais dos modelos 3D feitos no 3D Studio não são compatíveis com o Unreal Engine 4 e foram totalmente refeitos no *Game Engine* para se reproduzir a aparência da cena original pré-processada. O aspecto dos materiais que exigiu maior atenção consiste na simulação de reflexos**.** Na Abordagem de Pré-Processamento utilizam-se frequentemente materiais com reflexos calculados com a técnica Raytracing. Essa técnica simula o caminho percorrido pelos raios de luz e a forma como refletem nos objetos após interagirem com cada polígono de sua superfície. Apesar dos ótimos resultados proporcionados pelo Raytracing, seu uso é muito custoso para ser aplicado em um ambiente processado em tempo real, exigindo o uso de algumas técnicas alternativas. O Unreal Engine oferece duas técnicas distintas de reflexão de imagens. A primeira faz a captura de uma imagem panorâmica estática de 360 graus de um determinado local da cena e projeta essa imagem no objeto a refletir o ambiente. Essa opção tem bom desempenho, mas produz reflexos de baixa qualidade que não são ideais para objetos brilhantes como vidro e metal polido, por exemplo. A segunda utiliza uma ferramenta de captura que processa outro ponto de vista do ambiente em tempo real e o projeta no material desejado. Essa opção produz reflexos nítidos, mas causa um grande impacto no desempenho e deve ser usada de forma moderada. Na cena remodelada, esse tipo de reflexo foi usado em alguns objetos de vidro.
- **Iluminação:** No mundo real, os raios oriundos de uma fonte luminosa refletem nos objetos inúmeras vezes, perdendo energia durante cada interação. Essa propriedade causa um fenômeno conhecido como Iluminação Global, que faz com que a iluminação de um ambiente seja influenciada pelo próprio

ambiente, transmitindo cor e intensidade de luz de um objeto para outro. As imagens de Computação Gráfica deram um grande passo em rumo ao realismo quando passaram a simular esse efeito. Na Abordagem de Pré-Processamento, são gastas muitas horas de cálculos de iluminação com esse fim.

Existem algumas ferramentas recentes para simulação de Iluminação Global em tempo real em *Game Engines*. O Unreal Engine 4 possui sua solução de Iluminação Global em tempo real, chamada DFAO (*Distance Field Ambient Occlusion*). A Nvidia possui um *framework* chamado GameWorks que inclui uma série de bibliotecas com recursos gráficos que tiram proveito de instruções dedicadas em suas GPUs. Um desses recursos é uma técnica de simulação de Iluminação Global em tempo real chamada VXGI (*Voxel Global Illumination*) (NVIDIA, 2015d). O próprio Unreal Engine 4 possui sua solução de Iluminação Global em tempo real em faze de experimentação, chamada DFGI (*Distance Field Global Ilumination*), uma extensão de um recurso já presente oficialmente chamado DFAO (*Distance Field Ambient Occlusion)* (EPIC GAMES, 2015b). Por ainda não estar finalizado, o DFGI ainda possui comportamento imprevisível em determinadas situações, como granulação na imagem em ambientes escuros e vazamentos de luz em objetos com superfícies extensas.

A Nvidia possui sua própria técnica de Iluminação Global em tempo real chamada VXGI (*Voxel Global Illumination*)(NVIDIA, 2015d), disponível em seu *framework* para jogos GameWorks (NVIDIA, 2015e). Esse *framework* está em constante desenvolvimento e disponível gratuitamente para Unreal Engine 4. Por ser uma tecnologia nova, ainda não há documentação sobre como usá-la. As trocas de informações dependem da interação com os próprios desenvolvedores nos fóruns do *engine*. Foram feitos alguns testes de reconhecimento da ferramenta e os resultados visuais alcançados são promissores, apesar do custo de desempenho ser alto mesmo em um computador *desktop*[6] topo de linha.

Uma alternativa às técnicas de Iluminação Global em tempo real é o uso de *Lightmaps*. Tratam-se de texturas que contém informações de iluminação global pré-

[6] No momento da realização deste trabalho, configuração: Processador Intel Core i7, 16gb de RAM e placa de vídeo NVIDIA GTX980Ti 6gb de VRAM

processadas e são compostas com os materiais de cada objeto para se obter um resultado mais complexo do que os oferecidos pela iluminação em tempo real simples. Essas texturas fazem uso do canal de mapeamento extra adicionado no passo **Mapeamento para Iluminação,** citado anteriormente. Dessa forma, não há cálculo de iluminação global realizado em tempo real durante a execução do jogo, pois as informações de luz já estão armazenadas em texturas aplicadas nos materiais dos objetos. O ganho de desempenho é considerável (100 quadros por segundo no uso dos *Lightmaps* contra 30 quadros por segundo no uso do VXGI em testes realizados durante a remodelagem do ambiente usado neste trabalho), e a qualidade da iluminação é potencialmente superior ao alcançado pelo VXGI, pois pode-se dedicar mais tempo no pré-processamento[7] da cena para se alcançar resultados melhores. Por outro lado, a iluminação do ambiente perde potencial de interatividade. Na prática, isso significa que as luzes não podem ser totalmente apagadas ou ter sua posição alterada em tempo real, pois sua iluminação foi calculada anteriormente e está armazenada nos materiais dos objetos. A Figura 16 mostra o ambiente original, pré-processado no 3D Studio com Vray. A Figura 17 mostra o mesmo ambiente após todo o processo de remodelagem, processado em tempo real no Unreal Engine com uso da técnica de *Lightmaps.* A Figura 18 mostra o ambiente no Unreal Engine com a técnica de Iluminação Global em tempo real da Nvidia, VXGI. Nota-se que, enquanto que a técnica de *Lightmaps* apresenta uma qualidade muito próxima da imagem pré-processada, o VXGI possui qualidade levemente inferior, com menor definição nas sombras de contato entre objetos mais distantes. Isso se deve à aproximação que deve ser feita para viabilização da simulação de iluminação em tempo real. *Voxels,* no nome da técnica, vem do tipo de aproximação utilizado, no qual todos os objetos são aproximados a cubos de volume variável durante o cálculo da iluminação. Objetos mais distantes são substituídos por *voxels* maiores, o que causa perda de definição na iluminação.

[7] O pré-processamento de *lightmaps* difere da técnica de pré-processamento de geração de imagem 3D citadas em outros momentos deste texto. Nesse primeiro, o pré-processamento ocorre antes da compilação da aplicação 3D. Pode ser interpretado como um tempo extra para criação dos materiais durante o desenvolvimento do *software*, o que não interfere em sua capacidade de ser processado em tempo real após compilado.

Figura 16 - Imagem Pré-Processada no 3D Studio com Vray (Fonte: Site da Evermotion - http://www.evermotion.org/shop/show_product/archinteriors-vol-29/9520)

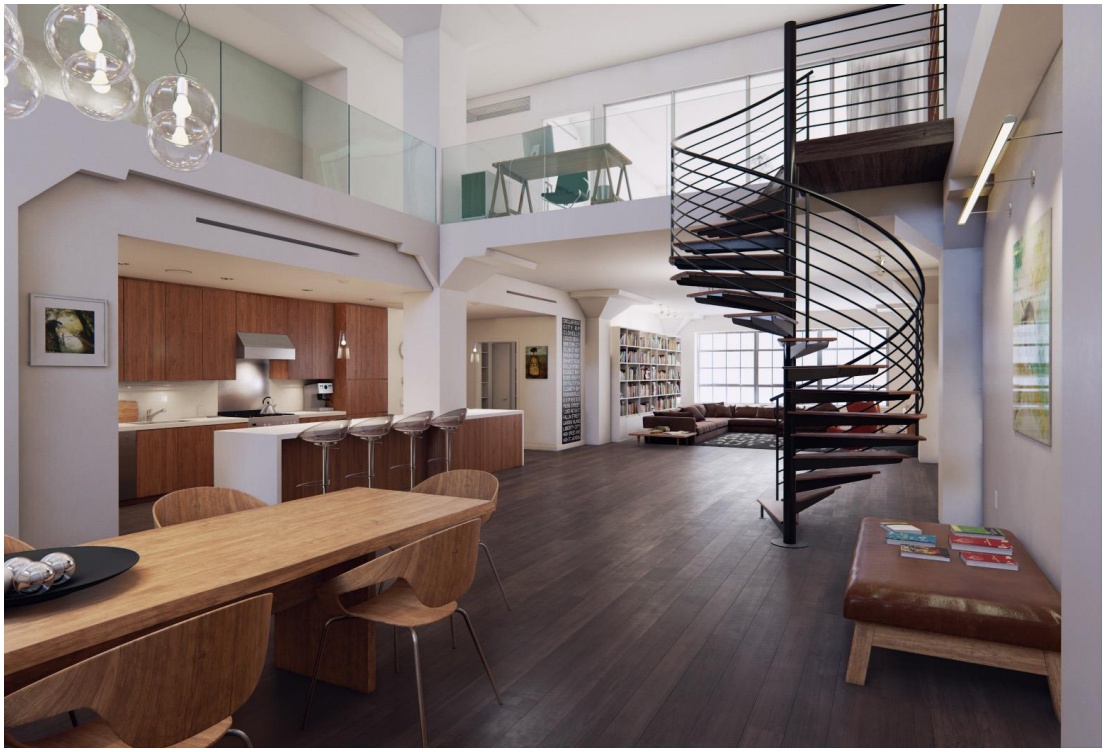

Figura 17 - Imagem Processada em Tempo Real no Unreal Engine 4 com uso da técnica de *Lightmaps* (Fonte: Próprio autor)

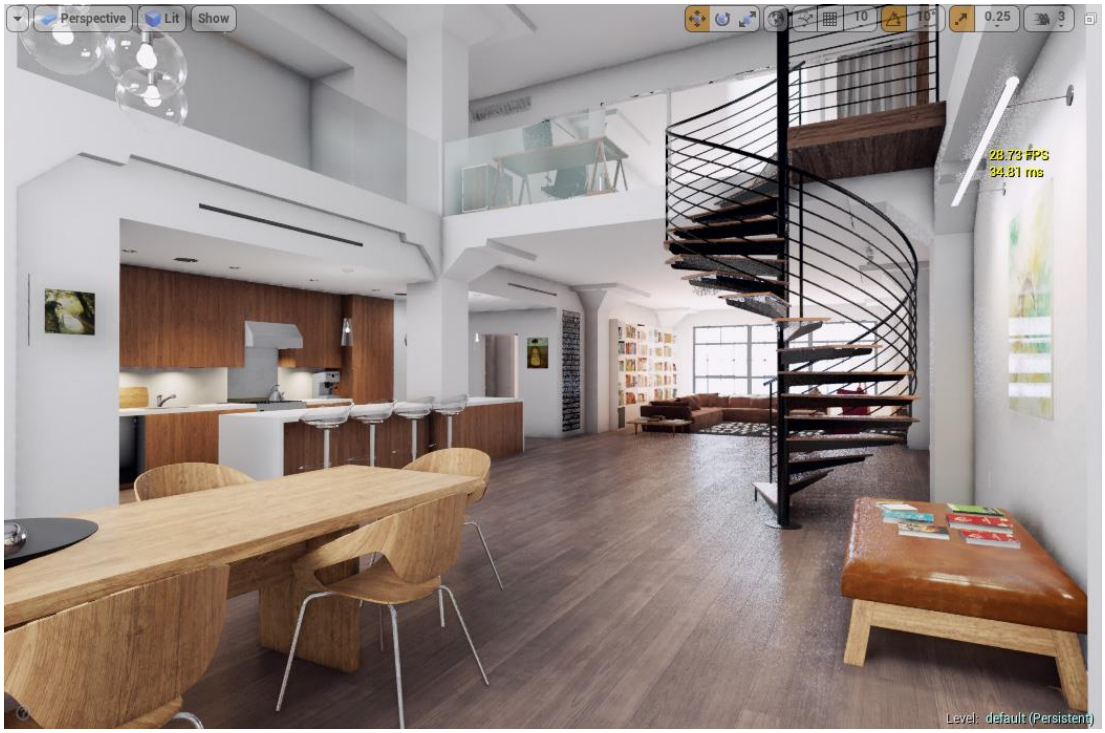

Figura 18 - Imagem Processada em Tempo Real no Unreal Engine 4 Com Uso da Técnica VXGI (Fonte: Próprio autor)

Para a HPM a ser utilizada nos testes da solução desenvolvida neste trabalho, optou-se pela técnica dos *Lightmaps* devido a configurações necessárias para compilação e testes do recurso VXGI que não puderam ser finalizadas a tempo. Se não fosse por esse motivo, a técnica de VXGI seria considerada a ideal por permitir maior grau de interatividade, característica importante de uma HPM.

4.4.3 Implementação da Lógica de Interação da HPM

A Lógica de Interação é análoga ao termo *Gameplay* da área de jogos. Trata-se de um conjunto de regras que define como o usuário poderá interagir com a aplicação 3D. No Unreal Engine, o *gameplay* deve ser programado preferencialmente com uso de classes básicas já oferecidas pelo *engine*. Caso não se use tais classes, perde-se a possibilidade de reuso de código otimizado já disponibilizado que possui integração com diversos recursos do motor gráfico. Essas classes, bem como todo o *engine* são implementadas em C++ e obedecem ao paradigma de Orientação a Objetos. As principais disponíveis no *gameplay* são:

- ***Actors:*** Classe genérica que representa qualquer objeto que pode ser colocado no ambiente 3D, ou *level*. Ela suporta transformações 3D como translação, rotação e escala e pode ser adicionada ou removida de um *level* em tempo real.
- ***Pawn:*** Herda da classe *Actor* e pode ser agente em um *level*. Isso significa que essa classe pode ser possuída por um *Controller* e interagir com outros *actors*.
- ***Character:*** Herda da classe *Pawn* e tem estilo humanoide. Possui por padrão um esqueleto humanoide ao qual podem ser anexados modelos 3D.
- ***Controller:*** Herda da classe *Actor* e é responsável por controlar um *Pawn*. Existem duas variações: ***PlayerController****,* que representa a interface entre um jogador humano e um *Pawn;* e ***AIController,*** que representa a interface entre um módulo de inteligência artificial e um *Pawn.*
- **HUD (*Heads-up Display*):** Classe que representa uma camada 2D que sobrepõe o conteúdo 3D mostrado na tela. Essa classe deve ser utilizada para a exibição de elementos gráficos como menus, ícones, textos, etc. Ela pode ser utilizada pelo *PlayerController*.
- ***PlayerCameraManager*:** Classe que representa a visão do jogador no ambiente 3D. Geralmente é utilizada pelo *PlayerController.*
- ***GameMode:*** Classe com definições do *gameplay* como regras e condições para se vencer um jogo.

O relacionamento entre essas classes é ilustrado na **Figura 19**.

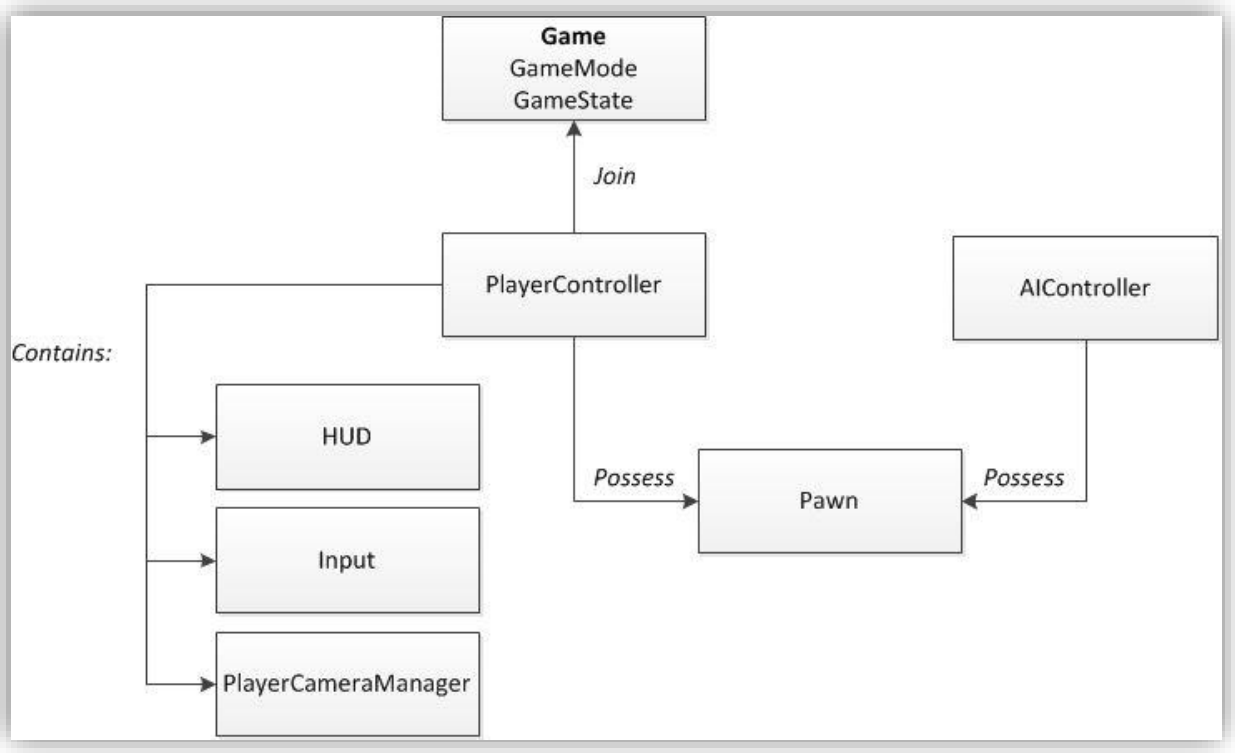

Figura 19 – Gráfico de relação entre classes do *gameplay.* (Fonte: Unreal Engine 4 Documentation - https://docs.unrealengine.com/latest/INT/Gameplay/Framework/QuickReference/index.html)

A HPM deste projeto consiste de um ambiente arquitetônico 3D, com visão em primeira pessoa, cuja modelagem foi abordada no item anterior, 4.4.2. Criou-se uma lista de interações que pudesse representar as principais possibilidades de interatividade com esse tipo de ambiente 3D:

- **Rotação da câmera**: É o controle que o usuário tem sobre a direção para a qual olha no ambiente virtual. A rotação pode ser no plano horizontal, vertical, ou em uma combinação de ambos.
- **Translação da câmera**: Trata-se do controle do usuário sobre a posição da câmera no cenário, movimentando-a como se caminhasse pelo ambiente. Pode ser feita para frente, para trás, para ou para os lados.
- **Seleção de objetos**: O usuário pode selecionar alguns objetos do cenário com os quais deseja interagir.

- **Edição, adição e remoção de objetos**: O usuário pode editar propriedades dos objetos selecionados como cor, tipo, posição e direção. Também pode remover o objeto selecionado ou adicionar novos ao ambiente.

Para a implementação dessas interações, o *gameplay* da HPM foi organizado com base nas classes básicas de *gameplay* do Unreal Engine apresentados anteriormente, mas com a customização de algumas delas como mostrado a seguir:

- ***BPRabbitCharacter***: Customização da classe padrão *Character* criada especialmente para a HPM. Nessa classe foi adicionada toda a lógica de interatividade com o cenário e comunicação com o Gerenciador de Comunicação. Essa classe possui duas variações: ***BPRabbitCharacterRabbit****,* que é utilizada na versão da HPM com o modo de interação para situações de latência e ***BPRabbitCharacterLocal****,* utilizada na versão que simula uma aplicação com interface comumente usada por aplicações semelhantes, sem a preocupação com latência.

  A lógica das interações implementadas nessa classe foi feita inteiramente através da ferramenta *Blueprint Editor* do Unreal Engine. Nessa ferramenta, as funções e variáveis da classe são criadas e organizadas de maneira visual através de gráficos. Se o programador necessita de uma funcionalidade de *gameplay* que não seja oferecida pelas funções de cada classe, ele pode criar novas funções também de forma visual, ou via código. Funções criadas em C++ podem ser expostas no *Blueprint Editor* para uso no ambiente visual do Unreal, possibilitando a combinação das duas formas de programação.

  Todas as funções de comunicação com o Gerenciador de Comunicação foram criadas em C++, com uso de uma biblioteca externa, e expostas no *Blueprint Editor.* Dessa forma, a implementação do *gameplay* fica contida no ambiente visual e independente dos recursos de comunicação de baixo nível da aplicação 3D.

- ***BaseCustomFurnitureBP:*** Customização da classe *Actor* com funcionalidades específicas de objetos do cenário com os quais o usuário pode interagir. A mobília do ambiente arquitetônico utiliza essa classe.

- ***BaseCustomHousepartBP:*** Customização da classe *Actor* com funcionalidades específicas de elementos do ambiente que podem ter sua textura modificada pelo usuário, mas não possuem outros tipos de interação. Paredes, piso, teto e outras partes fixas do ambiente utilizam essa classe.

Mesmo com cinco tipos básicos de interação utilizados no *gameplay* da HPM, a quantidade de gráficos gerados durante a implementação é considerável. A seguir é descrita a implementação da interatividade de seleção de objetos e o caminho que o fluxo de ações percorre em cada uma das classes envolvidas. As demais interações seguem o mesmo princípio.

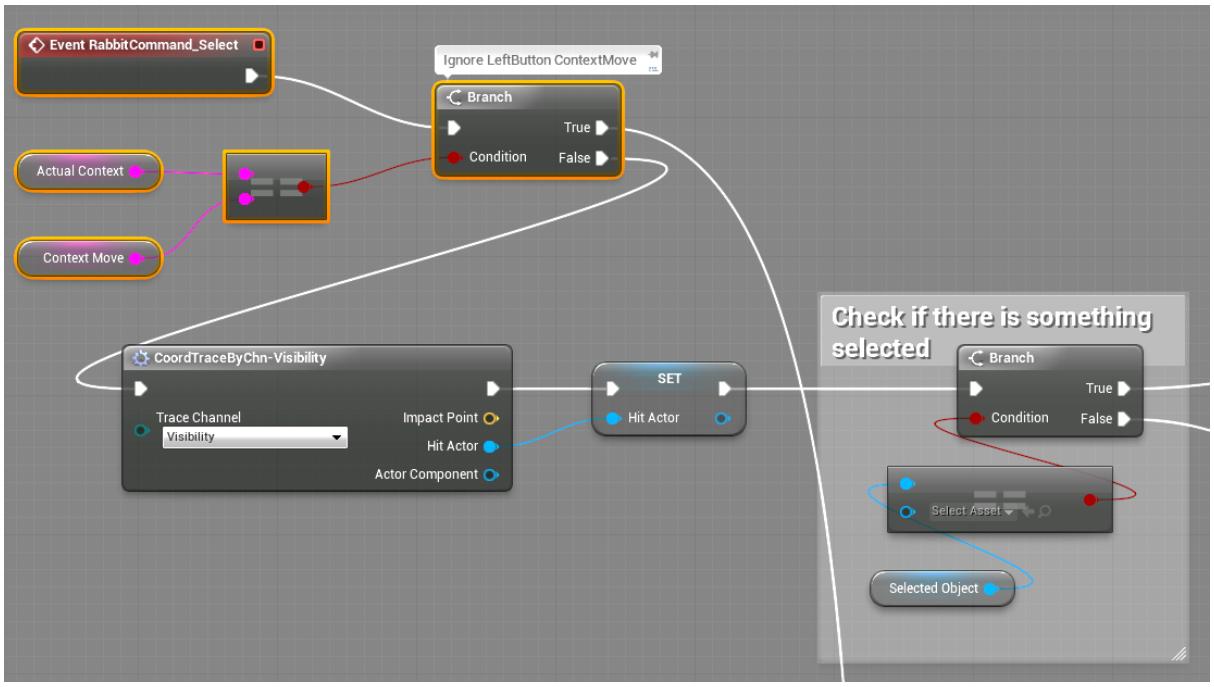

Figura 20 – *Blueprint 1 de* BP*RabbitCharacterRabbit:* Comando de seleção (Fonte: Próprio Autor**)**

Na Figura 20 é mostrada a primeira parte da lógica de interação na classe *BPRabbitCharacterRabbit*. O nó *Event RabbitCommand_Select* é a versão visual de uma função implementada em C++ responsável por ouvir o Gerenciador de Comunicação constantemente em espera de uma mensagem do Dispositivo Cliente que indique que um comando de seleção foi realizado. Assim que esse *input* é recebido, ocorre uma verificação do contexto do *gameplay* para decidir que tipo de interatividade deve ser realizada na aplicação 3D a partir do *input* de seleção. No

Dispositivo Cliente, a seleção ocorre através de um toque longo na tela sobre um objeto qualquer. Mas esse tipo de interação também é utilizado para definir a nova posição de um objeto que já foi selecionado e deve ser movido. Para decidir qual é a ação correta, definiu-se diferentes contextos de *gameplay* que mudam de acordo com o estado da aplicação. Se no momento do toque simples o usuário já havia selecionado um objeto e escolhido a opção de movê-lo, as sequências de *blueprint* que realizaram essas ações também alteraram o contexto da aplicação para *ContextMove*. Caso contrário, o contexto do *gameplay* será outro.

Há quatro contextos diferentes na lógica do *gameplay* da HPM: *ContextNavigation,* que representa a situação em que não há objetos selecionados e o usuário está navegando livremente pelo cenário; *ContextEdition*, que indica que um objeto foi selecionado; *ContextInsertion*, que é ativado quando o usuário deseja inserir um novo objeto no cenário; e *ContextMove,* que representa a situação em que o objeto selecionado está em modo de movimento.

Ainda na Figura 20, caso o contexto atual não seja *ContextMove,* um novo objeto deve ser selecionado. Para tanto, o nó *CoordTraceByChn-Visibility* dispara um raio de colisão a partir das coordenadas da tela em que o usuário tocou no Dispositivo Cliente. Há uma função implementada em C++ responsável por escutar o Gerenciador de Comunicações em espera de dados de coordenadas de cada toque que o usuário realiza na tela de seu dispositivo. Sempre que o *input* ocorrer, os dados da posição X e Y em porcentagem serão recebidos e atribuídos a variáveis globais correspondentes a cada coordenada na classe *BPRabbitCharacterRabbit.* Essas variáveis são acessadas pelo nó *CoordTraceByChn-Visibility* e qualquer outra função que necessite do posicionamento na tela. Se o raio disparado colidir com algum *Actor*, o mesmo é retornado para a variável *HitActor*. Em seguida, realiza-se uma verificação para determinar se já havia algum objeto selecionado no momento da nova tentativa de seleção. O fluxo lógico continua na Figura 21.

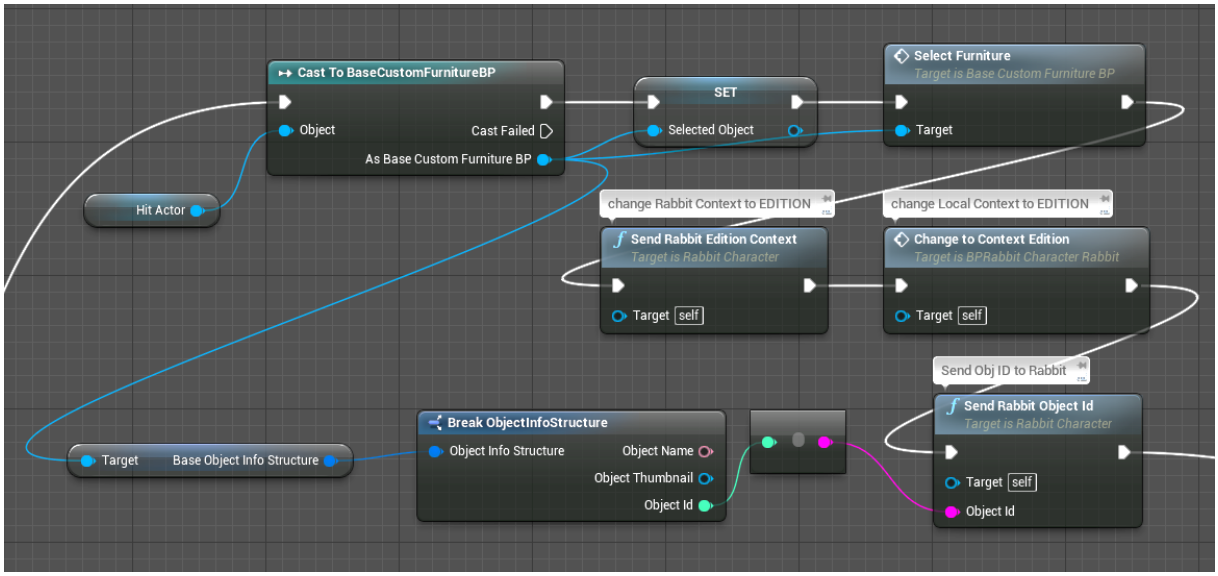

Figura 21 - *Blueprint 2 de* BP*RabbitCharacterRabbit:* Comando de seleção (Fonte: Próprio Autor**)**

Caso não haja objetos selecionados, é iniciada uma tentativa de execução de função em uma instância da classe *BaseCustomFurnitureBP.* As classes customizadas para os objetos do ambiente na HPM devem saber se estão selecionadas, além de realizar as funções de troca de materiais, movimentação e rotação. A tentativa é feita sobre o *HitActor*, que é uma variável de referência para objetos do tipo *Actor*. Como há outras classes nessa aplicação que herdam de Actor, a chamada de função de *BaseCustomFurnitureBP* pode falhar. Foi eliminada do gráfico da Figura 21 o fluxo que sai de *Cast Failed* no nó *Cast to BaseCustomFurnitureBP* para facilitar a visualização, mas na lógica do *gameplay* essa saída leva para a tentativa de *cast* em outras classes que possam ter sido selecionadas.

Se a tentativa for bem-sucedida, a variável *SelectedObject* recebe o valor de *HitActor*, que passará a ser um objeto selecionado. É então chamada a função *SelectFurniture* da instância de *BaseCustomFurnitureBP* que acabou de ser selecionada. Essa função permite que o objeto saiba que foi selecionado e tome as providências necessárias. Em seguida, a função customizada *SendRabbitEditionContext* é chamada e envia ao Gerenciador de Comunicação uma mensagem sobre o novo contexto do *gameplay ContextEdition*, para que a aplicação do dispositivo cliente tome as providências necessárias. *ChangetoContextEdition* altera o contexto do *gameplay* localmente. Por fim *SendRabbitObjectId* envia ao

broker o código de identificação do objeto que acabou de ser selecionado. Esse código é uma variável de uma estrutura da classe *BaseCustomFurnitureBP,* o que justifica os nós necessários no gráfico para obtenção da mesma.

Com os dados de contexto e identificação do objeto selecionado, a aplicação no Dispositivo Cliente é capaz de alterar seus menus locais para exibir opções referentes ao contexto atual da HPM. Se um objeto está selecionado, por exemplo, ao abrir o menu de opções o usuário terá acesso a configurações específicas daquele objeto. Se o contexto atual é o de navegação, a abertura do menu revelará opções específicas do ambiente em geral e da aplicação em si.

Na lógica do *gameplay* da HPM, um objeto selecionado sai da seleção caso o usuário tente selecioná-lo novamente ou tente selecionar qualquer outro objeto. Essa é a funcionalidade que segue da saída *False* da segunda comparação realizada na Figura 20. De forma análoga à seleção, é chamada uma função do objeto atualmente selecionado para indicar que ele deixou de estar selecionado e permitir que tome as providências adequadas. A sequência de nós que seguem da saída *True* da primeira comparação da Figura 20 trata da situação em que o contexto é *ContextMove.* Nesse caso, é disparado um segundo raio de colisão e o local de impacto é enviado para a instância do objeto selecionado através de sua função *MoveFurniture*, como mostrado na Figura 22.

O nó *ActorComponent* transmite ao objeto selecionado a informação de qual de suas partes foi selecionada. Essas partes específicas são elementos de interface que só aparecem quando um objeto está em modo de seleção. Tratam-se de setas de movimentação e rotação. Se nenhum desses objetos foi selecionado, significa que o usuário tocou sobre alguma posição do cenário para a qual pretende mover o objeto. Nesse caso, *BaseCustomFurnitureBP* utilizará o local de impacto *ImpactPoint* recebido como parâmetro em sua função *MoveFurniture* para transladar sua instância selecionada para este local. Caso uma seta de movimentação tenha sido selecionada, *BaseCustomFurnitureBP* ignorará o ponto de impacto e moverá sua instância em incrementos para a direção correspondente à seta. Se uma seta de rotação foi selecionada, o objeto sofrerá uma rotação incremental no sentido correspondente ao indicado por essa seta.

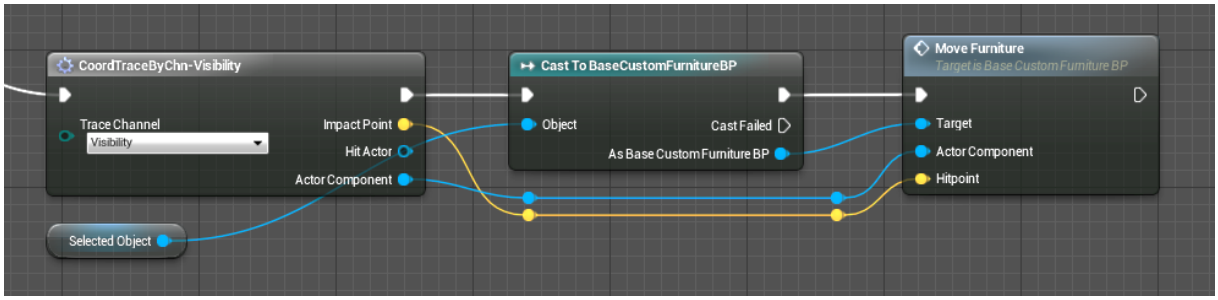

Figura 22 - *Blueprint 3 de* BP*RabbitCharacterRabbit:* Comando de seleção (Fonte: Próprio Autor**)**

Os pontos de impacto retornados pelas funções de *ray tracing* padrões do Unreal são justamente o que seu nome indica: coordenadas 3D do primeiro ponto com que um raio disparado a partir de coordenadas 2D específicas da tela colidiu no espaço. Com isso, se o usuário tocar sobre uma parede ou no teto, por exemplo, o objeto a ser movido seria simplesmente transladado para esses locais. Para impedir essa possibilidade, a classe *BaseCustomFurnitureBP* faz cálculos vetoriais sobre o ponto de impacto fornecido pela classe que a chamou. Do ponto de impacto, é calculado o vetor direcional que aponta para o jogador. Esse vetor é multiplicado por um escalar de valor equivalente a 50 centímetros e somado ao ponto de impacto, o que resulta em um ponto no espaço com 50 centímetros de distância do ponto de impacto em direção ao jogador. Essa distância evita que sejam gerados pontos de impacto muito próximo a superfícies que podem causar erros na detecção de colisão. A partir desse ponto é disparado um raio de colisão para baixo. O primeiro ponto de impacto retornado por esse raio é tomado como coordenada da nova posição do objeto. Na prática, essa solução garante que se o usuário seleciona regiões da parede ou do teto, o objeto é movido para a posição aproximada da projeção daquele ponto no plano do piso do cenário. A mesma técnica foi utilizada para cálculo da posição em que o próprio personagem se move durante o contexto de navegação. A Figura 23 mostra o *blueprint* dos cálculos citados acima.

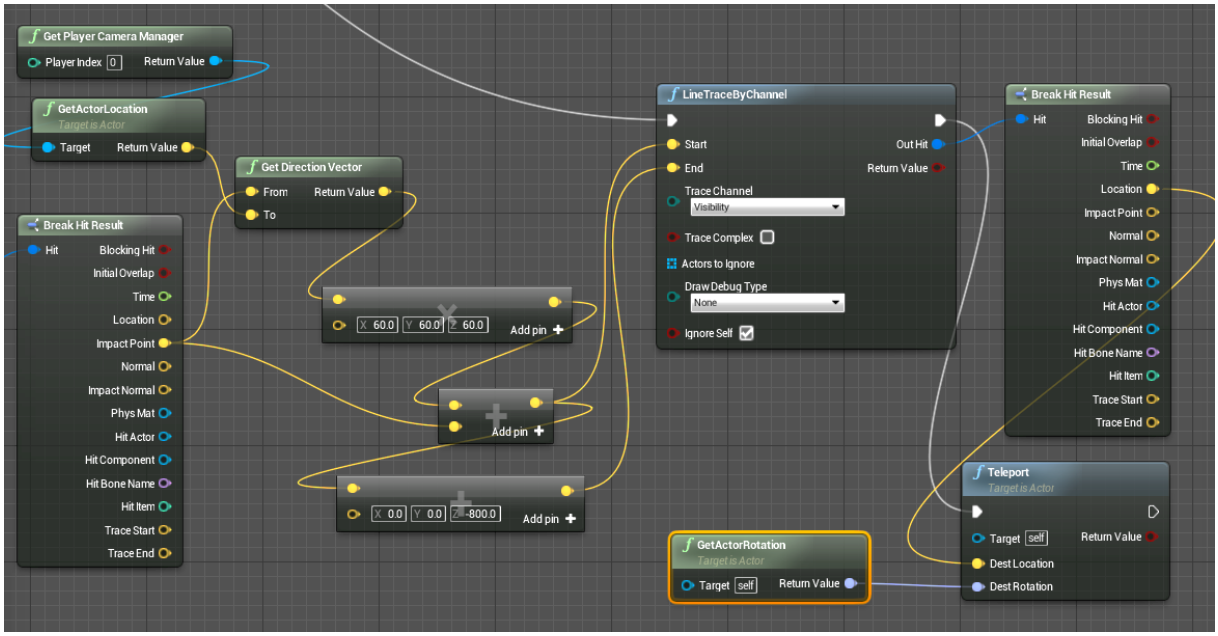

Figura 23 - *Blueprint 1* de *BaseCustomFurnitureBP:* Cálculo vetorial de posição de deslocamento (Fonte: Próprio Autor**)**

Duas funcionalidades do *gameplay* que exigiram atenção extra foram as interações de rotação e translação da câmera. A interação de rotação é determinada a partir de um toque simples em um ponto da tela. Toda vez que ocorre o *input* de um toque simples, a câmera deve realizar uma rotação que torne as coordenadas do toque o novo centro da tela. A Figura 24 mostra o *blueprint* desse mecanismo. O evento de *input* adequado ativa a função *CoordTraceByChn-Visibility* que dispara um raio de colisão que colide apenas com um tipo específico de superfície presente em uma esfera invisível de rotação fixa acoplada ao redor da cabeça do personagem. A colisão ocorre em um ponto do espaço e desse ponto gera-se um vetor de rotação através da função *FindLookatRotation*. Esse vetor possui os valores de rotação que devem ser aplicados na câmera para que ela centralize no ponto de impacto. Porém, essa rotação ocorre de forma brusca, o que causa um resultado visual desagradável. Para que a câmera gire suavemente até a nova posição, é necessário o uso de uma função de interpolação.

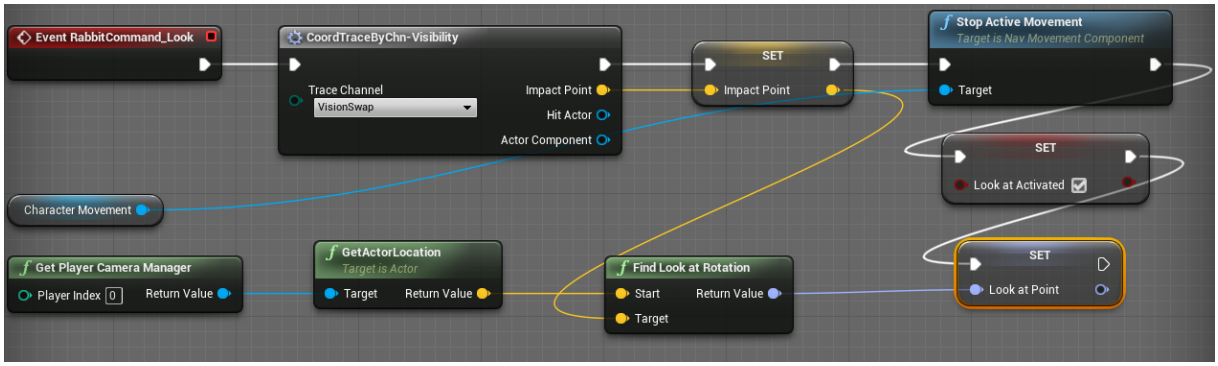

Figura 24 - *Blueprint 4 de* BP*RabbitCharacterRabbit:* Rotação da câmera (Fonte: próprio autor).

A translação da câmera é determinada a partir do toque duplo. No momento em que o usuário dá um toque duplo em alguma posição da tela, é detectado o ponto de impacto de um raio de colisão disparado a partir da coordenada do toque na tela. Então, o *BPRabbitCharacterRabbit* perde controle sobre o *Pawn* por um momento que passa a ser controlado momentaneamente por um módulo de inteligência artificial do *engine.* Esse módulo faz um cálculo para determinar qual é o menor caminho até o ponto de colisão sem que o *Pawn* colida com outros objetos durante o trajeto. Se esse caminho existir, o módulo movimenta a câmera até esse local através do caminho calculado. Assim que a posição desejada é atingida, *BPRabbitCharacterRabbit* retoma controle do personagem. Durante a translação, também é realizada uma rotação na câmera para que ela centralize no ponto de destino. Qualquer interação do usuário que ocorra durante a possessão do módulo de inteligência artificial causa a interrupção imediata da translação e retorna o controle do *Pawn* para o usuário.

Foram feitas duas versões da HPM, uma para a Solução Alternativa, e outra para a Solução Tradicional. As principais diferenças na lógica do *gameplay* entre as duas versões estão nos *inputs*, que são recebidos através da API do sistema para controle de *mouse* e *joystick*, e na lógica dos menus. Enquanto que na primeira versão a lógica dos menus ocorre no Dispositivo Cliente e é sincronizada através das mensagens de contexto de *gameplay* e id de objetos, na segunda versão os menus são processados e exibidos no servidor, como ocorreria em uma aplicação de *Cloud Gaming* comum.

4.4.4 Implementação das Funcionalidades de Comunicação da HPM e *Software* Gerenciador de Comunicação

Inicialmente, considerou-se o uso da própria API do *framework* Nvidia GameStream para comunicação entre cliente e servidor. Porém, a API é fechada e possui opções de envio de mensagens apenas no sentido cliente-servidor. No servidor, o GameStream mantém um serviço em execução que recebe as mensagens e as traduz em entradas de periféricos virtuais como *mouse*, teclado e *joystick*. Com funções personalizadas no Unreal Engine, foi possível emular interações típicas de dispositivos com *touchscreen*, como é o caso do cliente da arquitetura usada, através dos *inputs* de *mouse* e alavancas analógicas do *joystick.* Contudo, algumas das funcionalidades de interface propostas neste trabalho incluem elementos gráficos além do *streaming* de vídeo processadas localmente no dispositivo cliente. Esses elementos dependem de informações extras enviadas no sentido servidor-cliente, o que não é permitido pela API em questão.

Como possível alternativa, buscou-se por uma forma de troca de mensagens entre servidor e cliente através de um canal de comunicação independente do *framework* de *streaming* de vídeo. Uma opção seria o uso de funcionalidades de *socket* do Unreal. Porém, a biblioteca de *socket* nativa desse motor gráfico não é bem documentada e não existem classes nativas que proporcionem exemplos de sua implementação. No momento dessa etapa de desenvolvimento, o Unreal tinha pouco mais de um ano de existência e poucos trabalhos envolvendo comunicação via *socket* haviam sido desenvolvidos pela comunidade disposta a compartilhar a experiência. Optou-se então pelo uso de uma biblioteca externa que fornecesse boas funcionalidades de comunicação.

A alternativa considerada para essa função foi o *broker open source* RabbitMQ (RABBITMQ, 2015), que implementa o protocolo AMQP (*Advanced Message Queuing Protocol)*. A ferramenta fornece um servidor com suporte para diversas plataformas, dentre elas Windows, Debian e Fedora. Servidores EC2 da Amazon, também são suportados. As bibliotecas para cliente estão disponíveis oficialmente para Java, C# e Erlang, sendo diversas outras linguagens suportadas pela comunidade *open source*. Os testes com RabbitMQ acabaram gerando bons resultados e a ferramenta permaneceu como opção definitiva para a função de comunicação do projeto.

As plataformas utilizadas na arquitetura do projeto são Windows, para o servidor e Android para o dispositivo cliente. O servidor RabbitMQ foi hospedado no mesmo sistema da HPM. Essa escolha foi feita com a intenção de simplificar a estrutura da arquitetura e possibilitar maior controle sobre a latência para os testes. A biblioteca RabbitMQ cliente para C++ utilizada foi a SimpleAmqpClient, gerenciada pelo membro da comunidade *open source* Alan Antonuk. Para Android, utilizou-se o cliente oficial para Java. Todas as bibliotecas estão disponíveis no site oficial do RabbitMQ (RABBITMQ, 2015).

A adição da biblioteca SimpleAmqpClient no Unreal Engine 4 exigiu algumas configurações específicas nas configurações de *build* do projeto em C++ da HPM, pois o compilador utilizado pelo *engine* não é o mesmo utilizado pelo Visual Studio. Instruções para a adição e compilação de bibliotecas externas ainda não estavam documentadas oficialmente no momento da elaboração deste texto. O trecho de código necessário para configuração da biblioteca é apresentado na Figura 25

```
using UnrealBuildTool;
using System.IO;

public class Archinteriors29 : ModuleRules
{


    private string ModulePath
    {
        get { return Path.GetDirectoryName(RulesCompiler.GetModuleFilename(this.GetType().Name)); }
    }

    private string ThirdPartyPath
    {
        get { return Path.GetFullPath(Path.Combine(ModulePath, "../../ThirdParty/")); }
    }

    public Archinteriors29(TargetInfo Target)
    {
        PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore" });

        PrivateDependencyModuleNames.AddRange(new string[] {  });

        // third party includes
        PublicIncludePaths.Add(Path.Combine(ThirdPartyPath, "boost", "include"));
        PublicIncludePaths.Add(Path.Combine(ThirdPartyPath, "rabbitmqc", "include"));
        PublicIncludePaths.Add(Path.Combine(ThirdPartyPath, "SimpleAmqpClient", "include"));

        //third party libs
        PublicAdditionalLibraries.Add(Path.Combine(ThirdPartyPath, "rabbitmqc", "lib", "rabbitmq.4.lib"));
        PublicAdditionalLibraries.Add(Path.Combine(ThirdPartyPath, "SimpleAmqpClient", "lib", "SimpleAmqpClient.2.lib"));

        UEBuildConfiguration.bForceEnableExceptions = true;
    }
}
```

Figura 25 – Trecho de código do arquivo de configuração de *build* do projeto no Unreal Engine 4 para adição de bibliotecas externas. (Fonte: Próprio autor)

As pastas com os *headers* e as *libraries* das bibliotecas *SimpleAmqpClient* e suas dependências *rabbitmqc* e *boost*. Devem ser adicionadas na raiz da pasta do projeto. No exemplo acima, elas foram adicionadas em uma pasta chamada *ThirdParty*. As linhas de código abaixo dos comentários são as adições que carregam as bibliotecas externas. É importante que a variável *UEBuildConfiguration.bForceEnableExceptions* seja configurada como *true,* para que o tratamento de exceções das bibliotecas adicionadas funcione corretamente.

Devidamente adicionadas, a biblioteca pode ser utilizada para a elaboração de funções que se comuniquem com o *broker.* A lógica de comunicação foi totalmente implementada na classe *RabbitCharacter* em C++ e as funções de *input* e *output* foram expostas no *blueprint* para que pudessem ser utilizadas na implementação visual do *gameplay.*

A troca de mensagens no *RabbitMQ* utiliza filas. Um cliente do tipo *producer* envia uma mensagem para uma fila no *broker* e um cliente do tipo *consumer* recebe a mensagem dessa fila. Neste projeto, foi utilizado o modelo em que cada fila possui apenas um produtor e um consumidor. O produtor espera pela confirmação do *broker* de que sua mensagem foi entregue na fila e o *broker* espera por confirmação do *consumer* de que a mensagem foi devidamente recebida.

Para viabilização de todas as interações do *gameplay* da HPM, foram criadas uma série de filas e padrões de mensagens apresentados na Tabela 1.

FILAS		
Android -> UE4	**Mensagens**	**UE4 -> Android**
Aplicação		
QAU_APP_COMMAND	M_APP_COMMAND_EXIT	**QUA_APP_COMMAND**
QAU_APP_CONTEXT	M_APP_CONTEXT_NAVIGATION	**QUA_APP_CONTEXT**
	M_APP_CONTEXT_INSERTION	
	M_APP_CONTEXT_EDITION	
	M_APP_CONTEXT_MOVE	
Navegação		
QAU_NAV_COMMAND	M_NAV_COMMAND_LOOK [int 10]	
	M_NAV_COMMAND_WALK [int 20]	
	M_NAV_COMMAND_SELECT [int 30]	
QAU_NAV_X_COORD	[float 0 - 1]	
QAU_NAV_Y_COORD	[float 0 - 1]	

Objetos		
QAU_OBJ_ID	[string obj_id]	**QUA_OBJ_ID**
QAU_OBJ_COMMAND	M_OBJ_COMMAND_REMOVE	**QUA_OBJ_COMMAND**
	M_OBJ_COMMAND_MOVE	
	M_OBJ_COMMAND_ROTATE_CW	
	M_OBJ_COMMAND_ROTATE_CCW	
	M_OBJ_COMMAND_INSERT	
Materiais		
QAU_MAT_ID	[string mat_id]	

Tabela 1 - Filas e mensagens padrões utilizadas na comunicação entre Servidor e Cliente.

Nos nomes das filas, QAU (*Queue Android to Unreal*) e indica que o produtor da fila é sempre a aplicação cliente, enquanto que o produtor é a HPM. QUA (*Queue Unreal to Android*) tem o sentido contrário. As filas estão classificadas em: **Aplicação**, para troca de mensagens relacionadas ao contexto do *gameplay* e comandos da aplicação em si; **Navegação**, para mensagens relacionadas a rotação e translação da câmera; **Objetos**, para troca de mensagens que contenham comandos de movimentação e códigos de identificação de objetos; e **Materiais**, para envio de mensagens com código de identificação de materiais. Nem todas as filas possuem versões para os dois sentidos de comunicação.

Abaixo são apresentados trechos de código que ilustram a implementação de algumas das funções de comunicação expostas no *blueprint* e utilizadas na interação de Seleção apresentada no item 4.4.3.

```
void ARabbitCharacter::RabbitReceiveNavigationMsg()
{
    //Check new messages from QAU_NAV_X_COORD
    if (channel_receiver->BasicConsumeMessage(CONSUMER_NAV_X_COORD, nav_x_coord_envelope, 0))
    {
        FString converted_msg = ANSI_TO_TCHAR(nav_x_coord_envelope->Message()->Body().c_str());
        //GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red, converted_msg);
        x_mouse_coord = FCString::Atof(*converted_msg);
        this->MyBlueprintEvent_NewMousePosition(x_mouse_coord, y_mouse_coord); //MyBlueprintEvent_NewMousePosition
    }
```

Figura 26 – Trecho de código da classe *RabbitCharacter* com função de recepção das coordenadas do *input* do usuário.

A Figura 26 mostra parte de uma função que checa as mensagens disponíveis nas filas do tipo Navegação, segundo a Tabela 1. O objeto *channel_receiver* possui

um método capaz de consumir mensagens das filas do *broker*. Esse método é chamado dentro de uma condição *if* que só é ativada caso exista alguma mensagem na fila *CONSUMER_NAV_X_COORD*. Se o usuário tocou na tela de seu dispositivo, a aplicação cliente enviou uma mensagem com a coordenada x do toque para essa fila. Havendo mensagem, essa é convertida de *string* para *float* ao ser atribuída para a variável global *x_mouse_coord.* Essa variável é então passada como parâmetro para a função *MyBlueprintEvent_NewMousePosition,* que está exposta no *blueprint* da classe *RabbitCharacter* e suas derivadas em forma de um nó com *output* de valores *float* para as coordenadas X e Y mostrado na Figura 27.

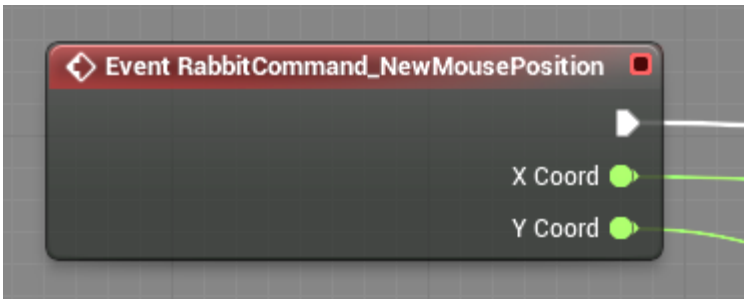

Figura 27 - *Blueprint 5 de* BP*RabbitCharacterRabbit:* Função *MyBlueprintEvent_NewMousePosition* (Fonte: Próprio autor)

```
void ARabbitCharacter::SendRabbitObjectId(const FString& objectId)
{
    string rabbitmsg(TCHAR_TO_UTF8(*objectId));
    msg_out->Body(rabbitmsg);
    channel_sender->BasicPublish("", QUA_OBJ_ID, msg_out);
}
```

Figura 28 - Trecho de código da classe *RabbitCharacter* com função de envio de ID de objeto. (Fonte: Próprio autor).

A Figura 28 mostra o código de uma função para envio de mensagem, *SendRabbitObjectId,* exposta no *blueprint* da Figura 21. Essa função recebe como parâmetro uma *FString*, classe de *strings* do Unreal, no fluxo de código visual no *blueprint* da classe *RabbitCharacter* ou suas derivadas. Essa variável é convertida para *string* e enviada para a fila *QUA_OBJ_ID* através de um método do objeto *channel_sender.*

A conexão com o *broker* é estabelecida toda vez que a HPM é inicializada. A partir desse momento, as funções que verificam a existência de mensagens nas filas são chamadas a cada 30 milissegundos. Por esse motivo, o recebimento de mensagens do *broker* está sujeito a uma adição de até 30 milissegundos na latência total da arquitetura. O envio de mensagens é instantâneo.

4.5 Construção da Aplicação Cliente

4.5.1 Escolha da Plataforma Móvel

Em meados de 2015, a plataforma Android representava cerca de 80% do mercado seguida pelo iOS com cerca de 15% e a tendência é de que esses valores se mantenham pelos próximos cinco anos (FORBES, 2015b) (IDC, 2015). Com esses dados, o Android pode ser considerado uma boa opção para se representar a plataforma móvel na solução desenvolvida neste trabalho. Além disso, as soluções escolhidas para codificação de *streaming* de vídeo e *broker* da arquitetura possuem bibliotecas oficiais para Android, o que facilita a implementação da Aplicação Cliente.

Pelas características da HPM produzida neste trabalho, acredita-se que *tablets* sejam mais adequados para sua visualização. Um ambiente 3D interativo com alta qualidade visual pode ter maior apelo se for explorado em uma tela maior do que a de *smartphones*. No Brasil, o uso de *tablets* tem sido impulsionado nas escolas como ferramentas de auxílio à educação (EDUCAÇÃO, 2014). A HPM pode ser usada com propósitos educativos, como na reprodução de museus, laboratórios virtuais e locais históricos, permitindo que o aluno explore e interaja com o ambiente. O processamento remoto, nesse caso, não só possibilita a reconstrução de aplicações visualmente fiéis ao mundo real, como também abre espaço para a implementação de simulação avançada de física em *hardware* especializado nos servidores. Considerando-se tais motivos, o *tablet* foi escolhido como tipo de dispositivo móvel na solução do trabalho.

A Aplicação Cliente foi pensada para uso em telas com mais de sete polegadas, não foi realizado trabalho para tornar a interface adaptável a tamanhos menores.

Porém, os conceitos utilizados na elaboração dos meios de interação para redução dos efeitos da latência podem ser aproveitados em outros tipos de interface.

A construção da Aplicação Cliente pode ser dividida em duas etapas: Implementação da Lógica de Interação e Implementação das Funcionalidades de Comunicação.

4.5.2 Implementação da Lógica de Interação da Aplicação Cliente

A Aplicação Cliente funciona como uma extensão da HPM processada remotamente. A solução de interatividade conta com formas específicas do usuário realizar ações no ambiente 3D e com menus processados localmente no dispositivo cliente para eliminação da latência em animações e interações com esses elementos.

Para a criação dos menus da aplicação, optou-se pelo uso do padrão *Navigation Drawer* da interface do Android (ANDROID, 2015). Esse padrão consiste de um menu que pode ser acessado com o deslizar do dedo a partir de uma das laterais da tela, normalmente o lado esquerdo. Se o menu não está sendo utilizado, fica escondido na lateral e libera área da tela para que a aplicação principal possa utilizá-la totalmente. Muitas aplicações para Android, como as oficiais do Google (GOOGLE, 2015), utilizam esse estilo de menu para maior aproveitamento da tela. Como as possíveis opções de interação da HPM dependem do contexto do *gameplay,* apresentados no item 4.4.3, optou-se por tornar o menu sensível a esses contextos ao invés de disponibilizar todas as interações possíveis de uma só vez. As versões de menu para cada contexto são ilustradas na Figura 29.

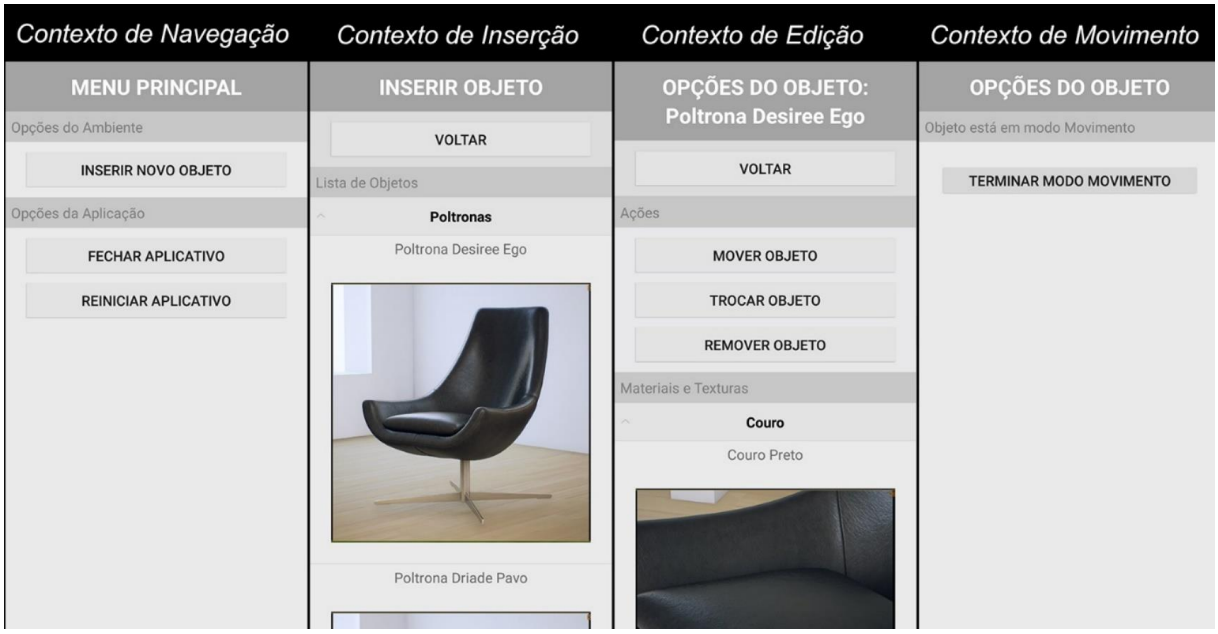

Figura 29 – Menus da aplicação categorizados por contexto de *gameplay*. Nomenclatura equivalente na HPM da esquerda para a direita: *ContextNavigation, ContextInsertion, ContextEdition, ContextMove* (Fonte: Próprio autor).

No Contexto de Navegação, nenhum objeto está selecionado e o usuário está navegando pelo ambiente. Se o menu for aberto neste momento, serão mostradas opções relacionadas ao ambiente 3D e à aplicação em si. Para a HPM desenvolvida, foram adicionadas as opções de Inserir Novo Objeto, Fechar Aplicativo e Reiniciar Aplicativo, mas aqui cabem opções como Mudar Ambiente, Salvar Ambiente, Capturar Imagem do Ambiente, Modo Multiplayer, etc.

Se o usuário seleciona a opção de Inserir Novo Objeto, esse comando é enviado à HPM remota, que mudará seu contexto e retornará uma mensagem à Aplicação Cliente indicando que o contexto foi alterado. Nesse momento, conteúdo do menu muda para se adequar ao Contexto de Inserção, com uma lista de objetos que podem ser inseridos e a opção Voltar, que retornará ao Contexto de Navegação. O mecanismo é semelhante para os demais menus, sendo que no Contexto de Edição a Aplicação Cliente recebe o identificador do objeto selecionado junto com a mensagem de mudança de contexto. O menu do Contexto de Movimento substitui o Menu de Edição quando o usuário seleciona a opção Mover Objeto. Na área sob o botão Terminar Modo Movimento podem ser inseridas opções de auxílio na movimentação para cada tipo de objeto selecionado como colisão com outros objetos, posicionamento em incrementos, atração por outros objetos próximos, limitações de área, etc.

Todo o conteúdo gráfico exibido nesses menus fica armazenado no próprio dispositivo. No cenário prático, as imagens dos objetos e materiais vêm incluídas na Aplicação Cliente desde sua instalação. Opcionalmente, pode-se realizar o *download* de pacotes com imagens de bibliotecas de objetos e materiais para cada ambiente 3D, desde que isso ocorra antes do início da navegação. O objetivo dessa escolha é garantir que as imagens estejam prontamente disponíveis no dispositivo quando o usuário interagir com os menus, sem tempos extras de carregamento. Quando a HPM envia uma mensagem de mudança de contexto e um identificador de objeto, a Aplicação Cliente busca pelos dados referentes àquele objeto em sua biblioteca local, sincronizada com a HPM em qualquer momento anterior ao início da navegação. A aplicação desenvolvida possui a biblioteca de imagens de cada objeto editável embutida no aplicativo, o que é suficiente para a realização dos testes.

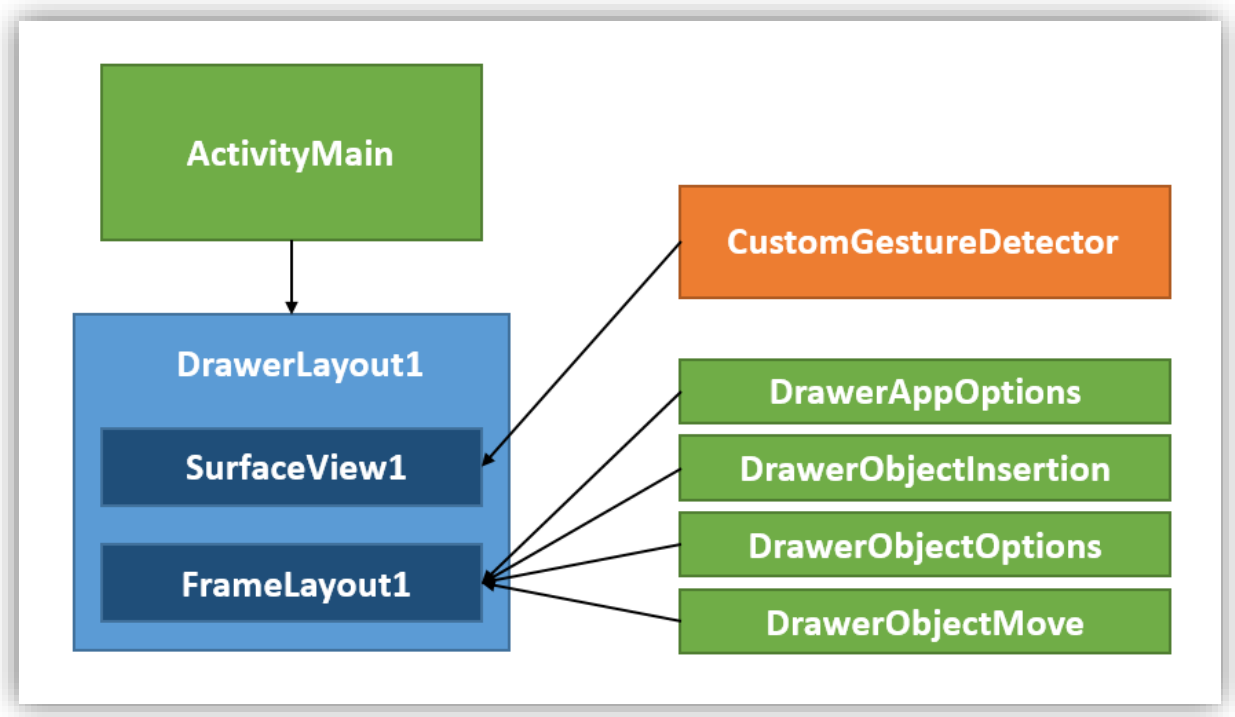

Figura 30 – Gráfico da organização simplificada do Aplicativo Cliente com suas principais classes e *layouts.* (Fonte: Próprio Autor)

A Figura 30 ilustra de forma simplificada a organização do aplicativo para Android. As direções das setas não simbolizam relações específicas entre classes, apenas ilustram como elas estão organizadas. A principal classe da aplicação é a *ActivityMain*. Essa classe é responsável pela comunicação com o Gerenciador de

Comunicação, reprodução do *streaming* de vídeo e troca dos menus de acordo com o contexto do *gameplay*. *DrawerLayout1* é o *layout* utilizado pela classe principal *ActivityMain*. Um *layout* contém definições em XML de estruturas que podem ser combinadas e aninhadas para exibição na tela. Essas estruturas podem ser manipuladas pela classe que utiliza o *layout* que as contem. No caso do *DrawerLayout1* existem duas estruturas que podem ser manipuladas pela *ActivityMain*: *SurfaceView1* e *FrameLayout1*. A primeira é associada ao *streaming* de vídeo e à classe responsável por interpretar as entradas de usuário para interação com o ambiente 3D, a *CustomGestureDetector*. A segunda, associa-se à classe responsável pelo menu referente ao contexto atual, que pode ser *DrawerAppOptions, DrawerObjectInsertion, DrawerObjectOptions* ou *DrawerObjectMove.* Essa estrutura do *DrawerLayout1* é padrão e deve ser respeitada para que a funcionalidade de deslize e sobreposição do menu lateral oferecido pela API do Android funcione adequadamente.

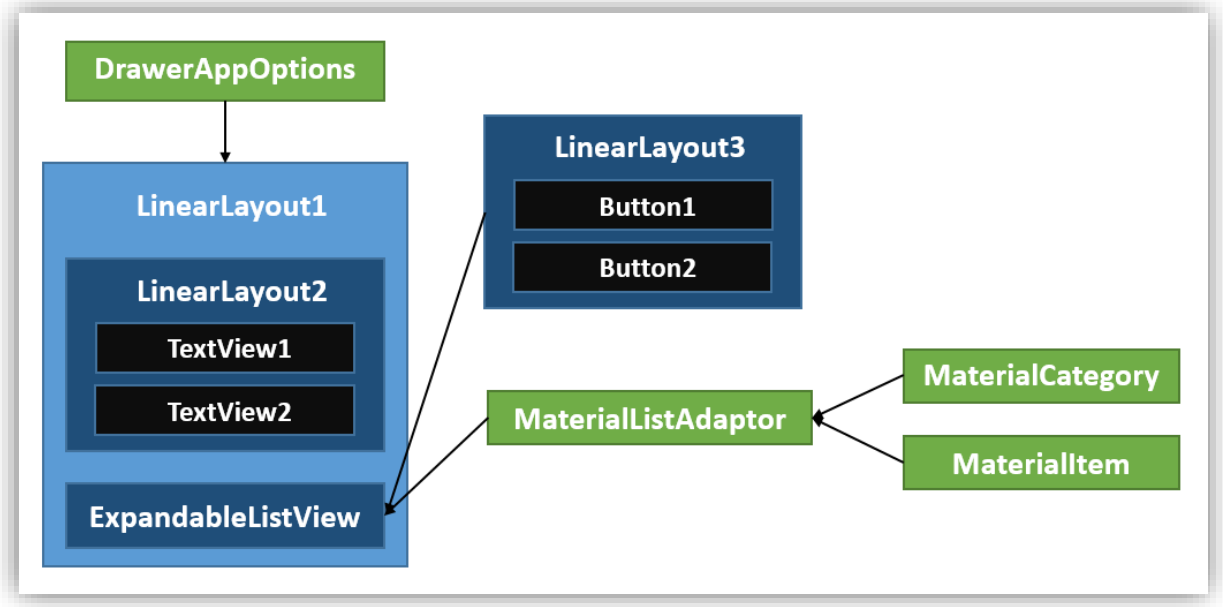

Figura 31 - Gráfico da organização simplificada do Menu de Edição do objeto e as classes e *layouts* com os quais se relaciona (Fonte: Próprio Autor)

Cada classe de menu utiliza um *layout* específico para exibir conteúdo na tela. A classe *DrawerAppOptions*, mostrada na Figura 31, utiliza um *LinearLayout1* com diversas estruturas aninhadas para organização dos itens do menu. *LinearLayout2* é usado para exibição do título do menu, em *TextView1*, e nome do objeto selecionado em *TextView2*. *ExpandableListView* é uma estrutura em forma de lista expansível na

qual itens podem ser adicionados conforme necessário. Se o objeto selecionado permitir interações de movimento, troca e remoção, é associado o *layout LinearLayout3* ao cabeçalho de *ExpandableListView.* Essa estrutura contém botões para realização das interações citadas. A classe *MaterialListAdaptor* é utilizada para o preenchimento de *ExpandableListView* com as opções de materiais do objeto selecionado. Cada material é representado por um objeto da classe *MaterialItem* e associado a uma instância da classe *MaterialCategory*. Todas essas associações são comandadas pela classe *DrawerAppOptions*, que se comunica com a classe *ActivityMain* para ter acesso às mensagens recebidas do Gerenciador de Comunicação e enviar mensagens novas, caso necessário.

As classes *MaterialItem* e *MaterialCategory* são basicamente estruturas com variáveis que definem esses elementos. No *gameplay* do Unreal, um material possui um **código identificador**, um **nome**, um **identificador de canal**[8], uma **categoria** e uma **imagem ilustrativa**. Todos esses componentes são replicados no Aplicativo Cliente através da classe *MaterialItem*. No momento em que *ActivityMain* instancia as classes de menus, são criadas listas de objetos do tipo *MaterialItem* os quais recebem valores para cada uma de suas variáveis. Esses valores são resgatados da biblioteca dentro do próprio aplicativo. *MaterialCategory* serve apenas para a organização dos materiais em categorias que podem ser expandidas no *ExpandableListView*.

4.5.3 Implementação das Funcionalidades de Comunicação da Aplicação Cliente

Da mesma forma que a HPM, a Aplicação Cliente faz uso de uma biblioteca cliente do RabbitMQ para comunicação com esse *broker.* O estabelecimento de uma conexão, monitoramento de filas para recebimento de mensagens e publicação de mensagens ficam a cargo da classe *ActivityMain*.

Assim que a aplicação é iniciada, a função *setupConnectionFactory*, responsável por estabelecer uma conexão com o *broker* é chamada. Em seguida, são

[8] Identificador de canal é um código que associa o material a uma parte do objeto 3D. Os objetos 3D podem ter diferentes tipos de materiais aplicados em cada um de seus polígonos como citado no item 4.4.2 - Modelagem do Ambiente 3D da HPM.

inicializadas três *threads*: *subscribeAppContext, subscribeObjId* e *createPublishQueue.* As duas primeiras checam se a conexão com o *broker* foi bem-sucedida, cria respectivamente as filas QUA_APP_CONTEXT e QUA_OBJ_ID, caso ainda não tenham sido criadas pela HPM e iniciam um *loop* de verificação de mensagens nessas filas. Dentro do *loop* são feitas verificações das mensagens recebidas para que se tome a atitude correta. Caso seja recebido uma mensagem de contexto, por exemplo, a variável global *new_context* recebe o valor dessa mensagem. Toda vez que o usuário solicitar a abertura do menu, é feita uma comparação entre essa variável e a *current_context*. Se elas forem diferentes, é realizada a troca do menu com base no novo contexto e *current_context* é atualizada. A terceira thread é responsável por criar todas as filas para publicação do cliente, caso ainda não tenham sido criadas pela HPM. Toda função que precisar publicar em alguma fila, o fará através do método *basicPublish* do objeto da classe *Channel* instanciado por essa *thread*, com o nome da fila e a mensagem passados como parâmetros.

A Aplicação Cliente da Solução Tradicional não possui nenhuma das funcionalidades de menus locais e comunicação com o *broker*. As entradas do usuário são capturadas de forma semelhante à de um Aplicativo Cliente de uma solução de *Cloud Gaming* comum e enviadas ao servidor através da API do *framework* da Nvidia disponibilizado no aplicativo MoonLight.

4.5.4 Implementação da Funcionalidade de *Streaming* da Aplicação Cliente

A biblioteca para Android utilizada para comunicação com o serviço do Nvidia GameStream executado no servidor é fornecida com o aplicativo *open source* MoonLight (MOONLIGHT, 2015). Esse aplicativo possui funcionalidades de conexão, envio de *inputs* do usuário e recebimento do *streaming* de vídeo gerado pelo *driver* de vídeo da Nvidia no servidor. Foi reaproveitado o código básico do aplicativo MoonLight para incorporação dessas funcionalidades no Aplicativo Cliente.

Através de uma API da Nvidia para envio de *inputs* ao servidor, o MoonLight permite que se envie mensagens de entrada de *mouse,* teclado e *joystick* para o serviço do Nvidia GameStream no servidor que utiliza *drivers* emuladores desses dispositivos para entregar as mensagens diretamente para o SO Windows, como

explicado no item 4.4.1. Apesar de não possuir um canal de retorno do servidor para o cliente, impossibilitando seu uso na Solução Alternativa, essa API pôde ser usada na Solução Tradicional, que se assemelha ao padrão de aplicações para *Cloud Gaming* para qual o aplicativo MoonLight foi originalmente desenvolvido. Na Solução Tradicional não existe o envio de mensagens do servidor para o cliente além do *streaming* de vídeo.

O estabelecimento de uma conexão entre a Aplicação Cliente e o servidor em cada uma das soluções possui as seguintes etapas:

- **Solução Tradicional:** 1) O serviço do Nvidia GameStream está em execução no servidor e foi previamente configurado com o caminho do executável de uma ou mais HPMs; 2) A Aplicação Cliente é inicializada no Dispositivo Cliente e a primeira tela possui um campo para inserção do IP de um servidor e uma lista de servidores previamente configurados; 3) A Aplicação Cliente requisita uma conexão com o servidor e o serviço do Nvidia GameStream realiza um processo de autenticação e estabelecimento da conexão com o cliente; 4) O serviço do Nvidia GameStream envia ao cliente uma lista de uma ou mais HPMs disponíveis para execução; 5) A Aplicação Cliente requisita ao servidor o início da HPM; 6) O serviço do Nvidia GameStream inicia o executável da HPM no servidor, começa a transmitir o *streaming* de vídeo para a Aplicação Cliente e receber mensagens de *input* que são entregues ao Windows através de *drivers* emuladores de *mouse*, teclado e *joystick.*
- **Solução Alternativa:** As etapas de 1 a 6 são idênticas às da Solução Tradicional; 7) A Aplicação Cliente e a HPM iniciam a conexão com o Gerenciador de Comunicação, que os vincula através das filas específicas requisitadas por cada um.

O Nvidia GameStream possui um recurso de compressão de vídeo que ajusta o *bitrate* do *streaming* automaticamente de acordo com a banda da conexão entre o servidor e o cliente. É possível utilizar a API do MoonLight para configurar valores de resolução, taxa de quadros por segundo e *bitrate* máximo para o vídeo.

AVALIAÇÃO

Neste capítulo é descrito o processo de avaliação deste trabalho. No item 5.1 são apresentados os objetivos e métodos utilizados. Em 5.2 são detalhados os questionários e tarefas realizadas pelos participantes dos testes. Em 5.3 é detalhada a configuração do sistema utilizado nas avaliações e em 5.4 são expostos e analisados os resultados.

5.1 Objetivos e Métodos Utilizados na Avaliação

O objetivo da etapa de avaliação do projeto é avaliar a qualidade da experiência da Solução Tradicional e da Solução Alternativa em situação de alta latência e obter a opinião dos usuários sobre a qualidade visual da HPM utilizada e sobre a capacidade da aplicação arquitetônica em cumprir suas principais funções. Para tanto, foi elaborado um teste com um conjunto de tarefas a ser realizado pelos usuários enquanto interage com a HPM através de cada uma das interfaces. Essas tarefas cobrem todos os tipos de interação descritos no item 4.3.1.

Para a geração de resultados comparáveis entre as duas Soluções, utilizou-se a métrica MOS (*Mean Opinion Score*) (ITU-T, 2013), bastante comum em testes subjetivos de qualidade de áudio e vídeo (STREIJL; WINKLER; HANDS, 2014). Ela é baseada na percepção humana e geralmente utiliza uma escala discreta de valores de 1 a 5 em questionários respondidos pelo usuário durante ou após a realização dos testes.

Também foram realizadas entrevistas com cada usuário para obtenção de suas opiniões sobre a aplicação, as interfaces, suas preferências, críticas e sugestões.

Foi convidado um total de 10 usuários para participação nos testes.

5.2 Conjunto de Tarefas e Questionário

Antes do início dos testes, o usuário recebeu instruções para se colocar na posição de um possível comprador de um imóvel que estaria conhecendo o local através da aplicação, com interesse em explorar o ambiente, observar detalhes da arquitetura e interagir com a decoração.

A Figura 32 mostra o ambiente 3D da HPM visto de cima. Ao iniciar a aplicação, a câmera estava posicionada no centro do mapa, próximo ao ponto 1. Os comandos da interface foram ensinados ao usuário, que ficou livre para interagir com o ambiente pelo tempo que achasse necessário para se familiarizar com a interface. O tempo de familiarização é importante para que se reduza a possibilidade da falta de prática do usuário interferir em sua avaliação de cada interface. Em seguida, ele recebeu uma lista de tarefas a ser cumprida com uma ou duas questões relacionadas a cada tarefa. Não foi imposto limite de tempo para realização das tarefas e o usuário ficou livre parar tirar possíveis dúvidas sobre a interface durante a realização das mesmas. Tanto o período de treinamento quanto o de realização das tarefas foram realizados com as aplicações sujeitas ao valor constante de 800 milissegundos de latência simulada. Esse valor está baseado nos valores de latência discutidos na sessão 2.2. O *bitrate* do vídeo foi limitado em 1Mbit/s e a resolução em 1280x720 com 30 quadros por segundo.

Todos os usuários testaram primeiro a aplicação da Solução Tradicional e depois a Solução Alternativa e não tinham informações sobre a forma como as interfaces foram implementadas ou se estavam ou não preparadas para situações de

latência. A lista de tarefas e questões foi a mesma para cada interface e é citada e explicada abaixo:

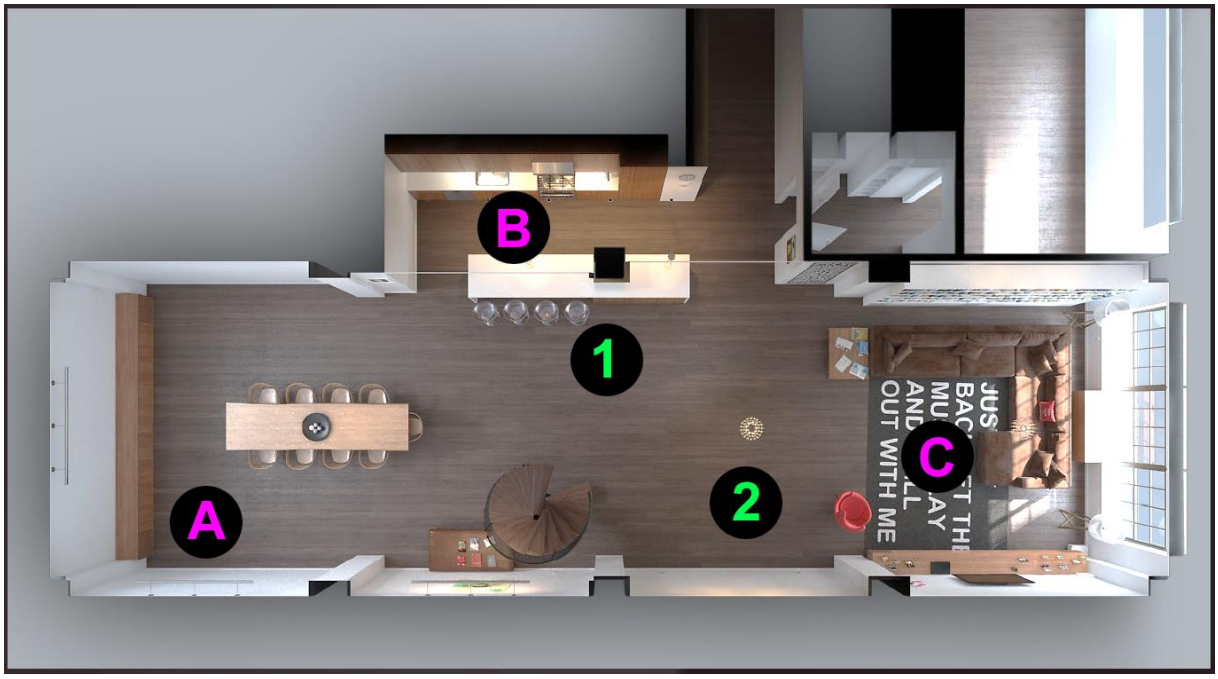

Figura 32 – Mapa do ambiente 3D com pontos de objetivos das tarefas (Fonte: próprio autor)

- **Tarefa 1:** “Avalie a facilidade da visualização de 360 graus do ambiente a partir do centro da sala, sem sair do lugar”.

Nessa atividade, o usuário realizou comandos de rotação da câmera e avaliou a facilidade que essa forma de interação oferece na observação geral do ambiente a partir de um mesmo local.

- **Tarefa 2:** “Avalie a facilidade de um passeio pelo ambiente buscando ter uma melhor visualização de todas as suas regiões”.

Essa atividade introduziu os comandos de translação da câmera na avaliação. Foi avaliada a facilidade que essa forma de interação proporciona em uma navegação geral pelo ambiente, para se observar diferentes regiões. A rotação foi usada de forma combinada com a translação.

- **Tarefa 3:** “Procure e observe de perto os seguintes objetos nessa ordem: quadro de flores rosas, fogão e lustres da mesa de jantar. Avalie a facilidade da observação desses detalhes específicos do ambiente”.

  Esses objetos estavam posicionados respectivamente nos pontos C, B e A da Figura 32. Com essa atividade, o usuário utilizou os comandos de rotação e translação da câmera para realizar movimentos mais precisos de forma a se locomover em espaços pequenos e se posicionar de maneira específica no ambiente.

- **Tarefa 4:** “Coloque uma nova poltrona de qualquer tipo no centro da sala e a posicione ao lado da poltrona já existente. Avalie a facilidade de inserção de um novo objeto no ambiente; avalie a facilidade de movimentação de objetos”.

  Essa tarefa foi subdividida em duas etapas. A **Tarefa 4A** consistiu na seleção e inserção do objeto e a **Tarefa 4B** na a movimentação desse objeto. O usuário fez duas avaliações. A questão sobre a facilidade de inserção de um objeto envolveu exclusivamente a manipulação de menus, uma vez que o objeto escolhido dentre as opções de uma lista aparece automaticamente na frente da câmera. A questão sobre a facilidade de movimentação do objeto avaliou as formas específicas para esse tipo de interação em cada uma das interfaces.

- **Tarefa 5:** “Altere a cor das paredes para vermelho. Avalie a facilidade de escolha mudança de cor do objeto”.

  A alteração de cor das paredes envolveu os comandos de seleção de objetos e interações com os menus para escolha e aplicação da cor desejada.

Esse conjunto de tarefas foi realizado para cada uma das interfaces da aplicação. E as respostas das avaliações utilizaram o sistema de escala do método *Mean Opinion Score* com cinco possíveis notas classificadas da seguinte forma: **1 –** Muito Ruim; **2 –** Ruim; **3 –** Razoável; **4 –** Bom; **5 –** Muito Bom.

Em seguida, o usuário respondeu questões gerais sobre a aplicação listadas a seguir:

- **Questão 1:** “Observe as imagens apresentadas pelo avaliador. Considerando que essas imagens são estáticas e demoram 4 horas para serem geradas em um computador de última geração, avalie a qualidade visual da aplicação gerada em tempo real que você acabou de testar”.

Nesse momento o usuário pôde observar e comparar uma imagem pré-processada nos *software* 3D Studio e Vray do mesmo ambiente arquitetônico da HPM, como mostrado na Figura 16 e na Figura 17. O objetivo dessa pergunta foi avaliar o grau de semelhança entre as imagens geradas em tempo real através das técnicas descritas 4.4.2 com imagens geradas por técnicas de pré-processamento na opinião do usuário.

- **Questão 2:** “Se você estivesse interessado na compra de um imóvel, como avaliaria a utilidade dessa aplicação? ”

Com essa pergunta, buscou-se obter a opinião do usuário sobre a utilidade da ferramenta em sua aplicação prática, com qualquer uma das interfaces.

- **Questão 3:** “Qual sua habilidade de interação com jogos em primeira pessoa? ”

Essa questão visou a detecção da possível relação entre os resultados das avaliações anteriores com o grau de habilidade do usuário com aplicações que empregam formas de interação semelhantes aos da HPM utilizada neste trabalho.

5.3 Configuração do Sistema de Testes

Para realização dos testes, foi utilizada a seguinte configuração de *hardware* e *software*:

- **Servidor:** Um computador *desktop* com processador Intel Core i7 3770 3.4 GHz, 16 gigabytes de memória RAM, uma placa de vídeo Nvidia GeForce 980Ti com 6 gigabytes de memória VRAM e sistema

operacional Microsoft Windows 10. Esse computador possui instalados os dois componentes de *software* relativos às duas versões de interface da aplicação, hospeda um *broker* RabbitMQ e executa os serviços da Nvidia relacionados à ferramenta GameStream.

- **Dispositivo Cliente:** Um *tablet* Samsung Galaxy Tab S de 10,5 polegadas, modelo SM-T800 com sistema operacional Android 5.0.1. Esse dispositivo possui instalados dois aplicativos relativos às duas versões de interface da aplicação.
- **Roteador:** D-Link modelo Cloud DIR-850L Gigabit Wireless AC1200 Dual Band.

O servidor foi conectado ao roteador via conexão Ethernet e o cliente via conexão WiFi de 5 GHz. Foi utilizada a ferramenta Clumsy 0.2 (JAGT, 2016) no servidor para simulação de latência de 800 milissegundos entre a conexão do servidor com o cliente. Esse valor foi determinado para representar uma situação crítica de conexão móvel com base em valores citados no item 2.2.

5.4 Resultados e Avaliação

Em todos os gráficos e tabelas dessa sessão, Aplicação 1 ou Série 1 referem-se à primeira aplicação testada pelos usuários, que corresponde à Solução Tradicional. Aplicação 2 ou Série 2 referem-se à segunda aplicação testada, correspondente à Solução Alternativa. Os gráficos da Figura 33 mostram as notas de 0 a 5 no eixo vertical para cada uma das tarefas por usuário. Série 1, com hachura, representa a Solução Tradicional e Série 2, sem hachura, representa a Solução Alternativa. No gráfico das Questões Gerais, cada questão é representada por uma cor de acordo com a legenda.

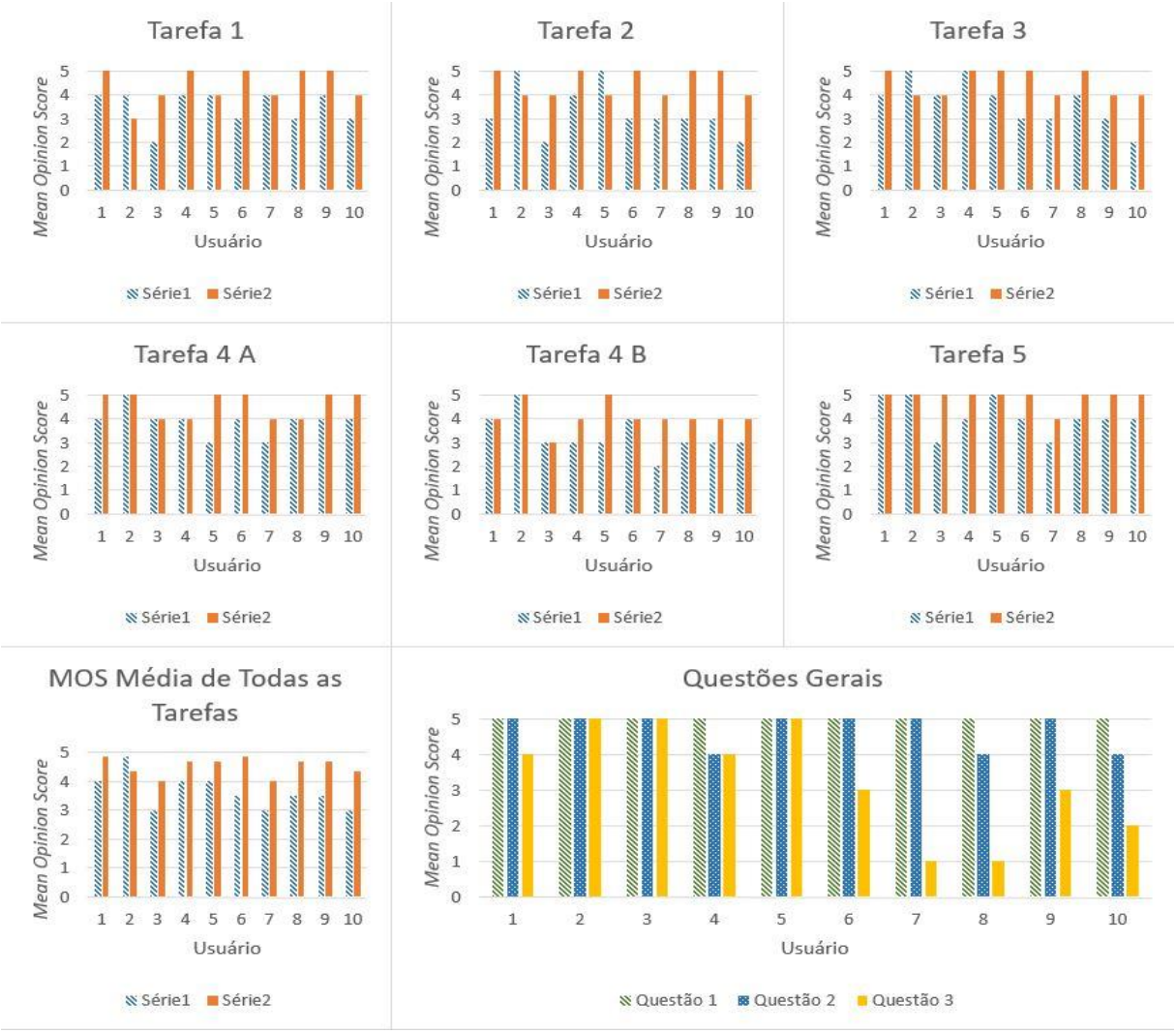

Figura 33 - Gráficos dos Testes de Usuário para cada Tarefa e Questão. Série 1 é relativa à Interface da Solução Tradicional e Série 2 à Solução Alternativa. (Fonte: Próprio Autor).

É possível observar pelos gráficos que a Solução Alternativa obteve notas iguais ou melhores às da Solução Tradicional de forma quase unânime em todas as tarefas. A exceção foi para o Usuário 2, que avaliou as tarefas de 1 a 3 com melhores nota na Solução Tradicional. Essas tarefas estão relacionadas às formas de interação para rotação e translação da câmera. Em entrevista, esse mesmo usuário declarou ter preferido o comportamento da Solução Alternativa no geral, mas que a quantidade de interações necessárias para realizar rotações da câmera em relação à Solução Tradicional não o agradou. Os demais usuários ressaltaram em entrevista a preferência pela Solução Alternativa. Abaixo são citados alguns comentários reforçam essa posição, sendo que a “primeira aplicação” se refere à Solução Tradicional, e a “segunda aplicação” à Solução Alternativa:

- **Usuário 3:** “A primeira solução foi difícil, bem mais difícil que a segunda. Na primeira dava até mais receio de explorar...”
- **Usuário 4:** “Na primeira é difícil de passar por lugares com pouco espaço, não tem precisão. A segunda é melhor, mais objetiva”.
- **Usuário 5:** “A tarefa para encontrar os objetos dá muito mais trabalho na primeira. O menu da segunda ficou muito melhor para mexer”.

Estudos indicam que, em testes subjetivos de qualidade de vídeo, cerca de 10 são o suficiente para se gerar alguns dados estatísticos (WINKLER, 2009). Apoiando-se nesse estudo, aplicou-se o Teste t de *Student* para a hipótese das amostras de médias MOS de todas as tarefas por usuário de cada aplicação serem iguais. Esses dados estão na Tabela 2 abaixo:

	MOS Médio App 1	MOS Médio App 2
Participante 1	4	4,8
Participante 2	4,8	4,3
Participante 3	3	4
Participante 4	4	4,7
Participante 5	4	4,7
Participante 6	3,5	4,8
Participante 7	3	4
Participante 8	3,5	4,7
Participante 9	3,5	4,7
Participante 10	3	4,3

Tabela 2 - Média de MOS de todas as tarefas por usuário para cada aplicação. A hipótese é a de que não há diferenças entre as amostras MOS Médio App1 e MOS Médio App2

Os cálculos resultaram em *p-value* = 0,0011. Considerando-se um nível de significância α=0,05, tem-se que *p-value* < 0,05. Pode-se então rejeitar a hipótese das amostras serem iguais e afirmar com 95% de confiança que existem diferenças significativas entre as médias de MOS para cada aplicação. O Teste t também permite que se rejeite com 95% de certeza a hipótese de que as amostras de respostas para cada tarefa são iguais entre aplicações diferentes, pois todos os *p-value* são menores que 5%, como mostrado na Tabela 3.

Aplicação 1 (Solução Tradicional)						
	Tarefa 1A	**Tarefa 2A**	**Tarefa 3A**	**Tarefa 4A**	**Tarefa 5A**	**Tarefa 6A**

Participante 1	**4**	**3**	**4**	**4**	**4**	**5**
Participante 2	**4**	**5**	**5**	**5**	**5**	**5**
Participante 3	**2**	**2**	**4**	**4**	**3**	**3**
Participante 4	**4**	**4**	**5**	**4**	**3**	**4**
Participante 5	**4**	**5**	**4**	**3**	**3**	**5**
Participante 6	**3**	**3**	**3**	**4**	**4**	**4**
Participante 7	**4**	**3**	**3**	**3**	**2**	**3**
Participante 8	**3**	**3**	**4**	**4**	**3**	**4**
Participante 9	**4**	**3**	**3**	**4**	**3**	**4**
Participante 10	**3**	**2**	**2**	**4**	**3**	**4**
Aplicação 2 (Solução Alternativa)						
	Tarefa 1B	**Tarefa 2B**	**Tarefa 3B**	**Tarefa 4B**	**Tarefa 5B**	**Tarefa 6B**
Participante 1	**5**	**5**	**5**	**5**	**4**	**5**
Participante 2	**3**	**4**	**4**	**5**	**5**	**5**
Participante 3	**4**	**4**	**4**	**4**	**3**	**5**
Participante 4	**5**	**5**	**5**	**4**	**4**	**5**
Participante 5	**4**	**4**	**5**	**5**	**5**	**5**
Participante 6	**5**	**5**	**5**	**5**	**4**	**5**
Participante 7	**4**	**4**	**4**	**4**	**4**	**4**
Participante 8	**5**	**5**	**5**	**4**	**4**	**5**
Participante 9	**5**	**5**	**4**	**5**	**4**	**5**
Participante 10	**4**	**4**	**4**	**5**	**4**	**5**
p-value	**0,01036169**	**0,00675671**	**0,03512374**	**0,00993242**	**0,02230883**	**0,00820318**

Tabela 3 - Teste t entre amostras de cada participante para uma mesma tarefa entre aplicações diferentes.

O gráfico da Figura 34 compara o MOS médio de cada tarefa por aplicação, incluindo o desvio padrão.

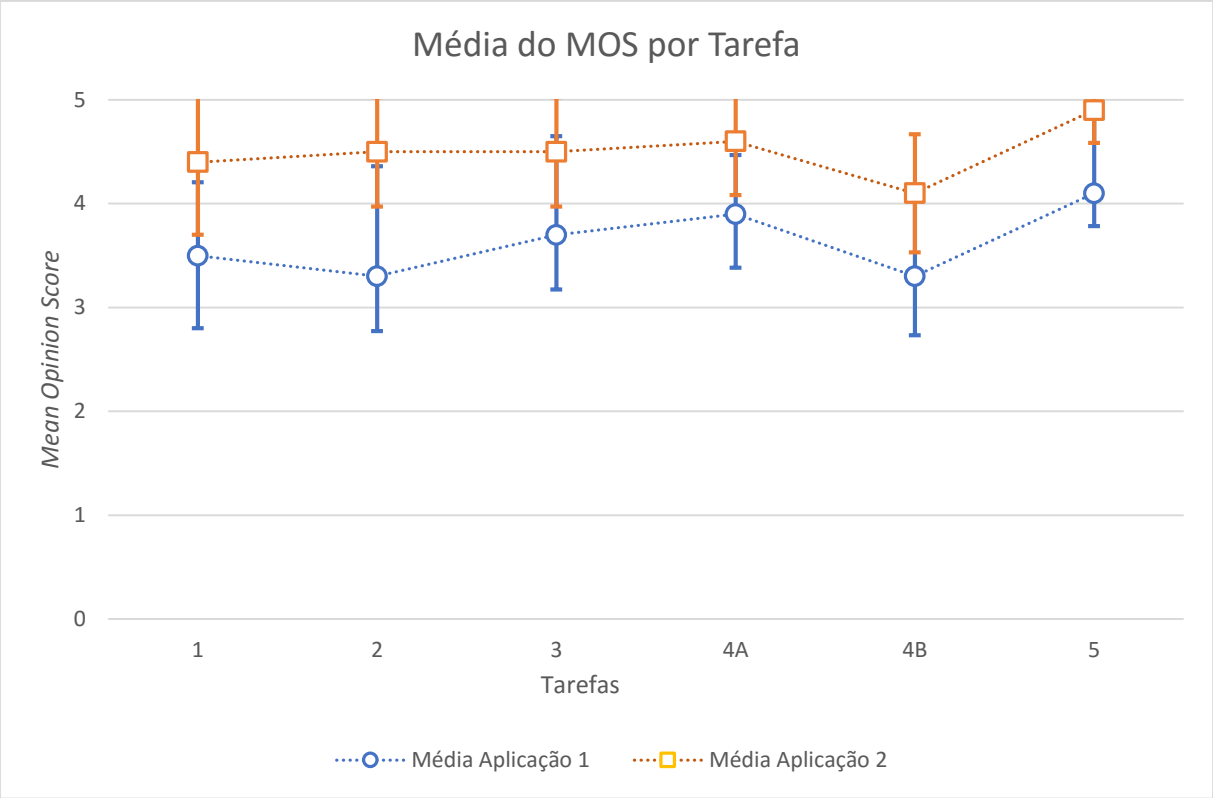

Figura 34 - Tabela comparativa da média de MOS entre as aplicações por tarefa.
(Fonte: Próprio autor)

O gráfico das Questões Gerais, na Figura 33, mostra que todos os usuários consideraram as imagens da aplicação semelhantes às imagens que utilizam a técnica de pré-processamento. Isso indica que as técnicas de remodelagem utilizadas neste projeto são uma boa opção para a substituição de mídias pré-processadas por HPM com processamento remoto sem perdas significativas de qualidade visual. A maioria dos usuários também considera que a aplicação cumpre bem sua função de visualização arquitetônica através das respostas da Questão 2.

A Questão 3 apresentou certa variação nos resultados. O gráfico da Figura 35 indica que existe uma tendência de que usuários com menor experiência em jogos de primeira pessoa deem notas inferiores para a Aplicação 1 em relação aos usuários com mais experiência. Porém, a amostra de usuários é muito pequena para que essa relação seja comprovada estatisticamente.

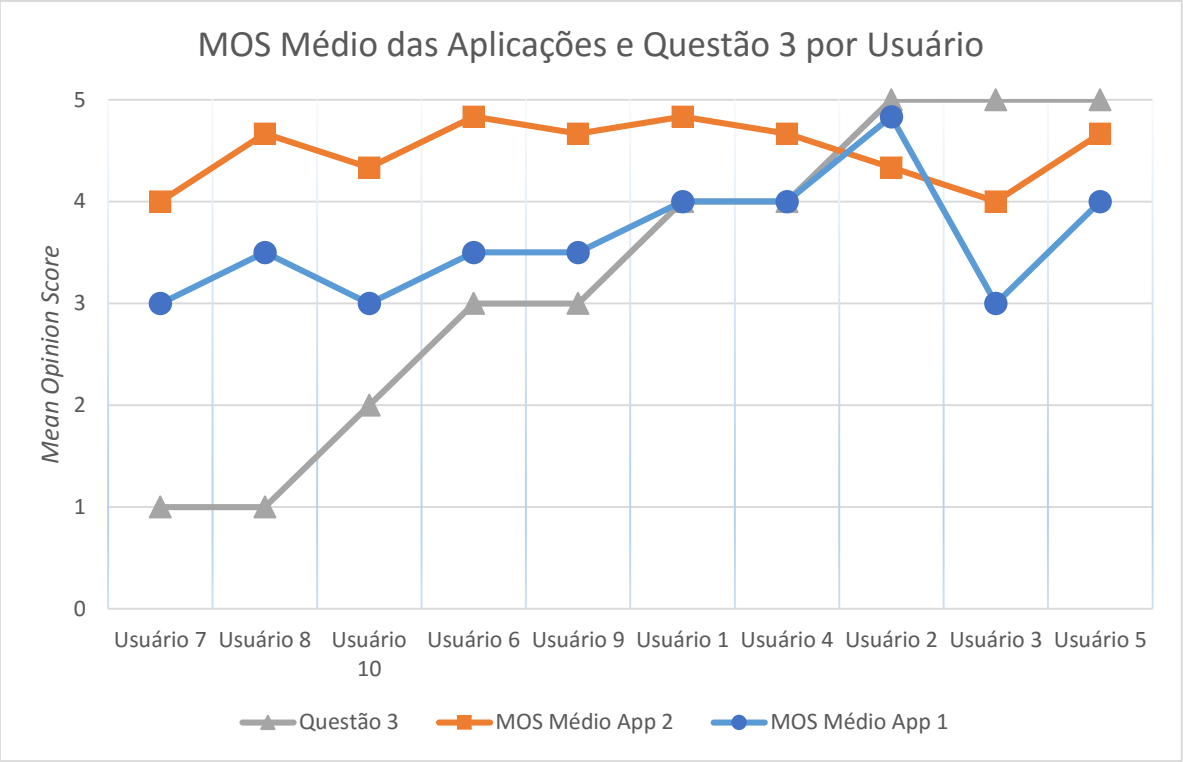

Figura 35 – MOS Médio das Aplicações e MOS da Questão 3 por Usuário. O eixo horizontal está em ordem crescente de MOS da Questão 3. (Fonte: Próprio autor)

Apesar da vantagem da Solução Alternativa em relação à Solução Tradicional indicada nos testes, os usuários fizeram considerações importantes durante as entrevistas. Uma delas é sobre a Solução Tradicional ser mais intuitiva:

- **Usuário 1:** “A primeira solução é muito prejudicada pelo atraso, mas é muito mais intuitiva. A segunda é melhor, mas demora mais para aprender, precisa de um tutorial”.
- **Usuário 5:** “Virar a câmera é mais intuitivo no primeiro, mas como tem atraso, fica difícil...”
- **Usuário 6:** “A segunda demora mais para memorizar, mas depois que me acostumei, ficou fácil”

Essas colocações fazem sentido, uma vez que a Solução Tradicional foi criada com base nos tipos de interações usados nos jogos e aplicações 3D mais famosos para dispositivos Android. Consequentemente, os usuários tendem a ter maior facilidade por já estarem acostumados a usar formas de interações semelhantes.

Os usuários utilizaram cada aplicação por cerca de 15 minutos. Os que notaram o maior tempo de aprendizado necessário na Solução Alternativa também disseram

ter se acostumado rapidamente. Isso indica que um bom sistema de tutorial e ajuda no aplicativo pode contribuir para uma curva de aprendizado ainda mais rápida.

Outro aspecto negativo da Solução Alternativa notado por alguns usuários refere-se ao recurso de rotação dos objetos, que permite apenas giros incrementais de 45 graus.

- **Usuário 1:** “A rotação de objetos na segunda solução é um pouco prejudicada pelos incrementos serem muito grandes, mas incrementos menores deixariam a rotação mais lenta”.
- **Usuário 5:** “Gostei mais da primeira para girar os objetos porque tem mais liberdade”.

A rotação incremental foi adotada para reduzir o impacto do *feedback* atrasado do movimento contínuo empregado na Solução Tradicional, mas ela cria um *trade-off.* Se os incrementos forem grandes, o processo de rotação é mais rápido, mas pouco preciso. Se forem pequenos, a precisão aumenta, mas rotações maiores exigirão muitas interações.

Alguns usuários notaram que, após algum tempo de uso, é possível se acostumar com a lentidão da Solução Tradicional e seu uso torna-se menos frustrante.

- **Usuário 6:** “Depois que a gente se acostuma com a lerdeza da primeira, até que fica fácil de controlar”.
- **Usuário 8:** “Dá para se acostumar com o atraso da primeira aplicação”.

A latência utilizada nos testes é de 800 milissegundos, o que representa um caso extremo se comparado com a média da latência de conexões móveis. Se for utilizada uma conexão de banda larga com WiFi, é possível reduzir essa média. Apesar de não ter sido realizado conjuntos de testes com valores de latência mais baixos, acredita-se que a diferença de preferência do usuário entre as duas soluções de interface seja menor. Uma alternativa interessante para se adequar a aplicação a diferentes cenários de latência seria possibilitar que o usuário escolha a interface que mais lhe agrada, ou mesmo permitir que se combine formas de interação das duas soluções. Um dos usuários chegou a sugerir que se a Solução Tradicional incluísse a forma de translação da câmera utilizada na Solução Alternativa, por meio de toque longo e movimentação automática, seria a alternativa ideal.

Capítulo 6

CONCLUSÃO

Este capítulo apresenta a conclusão do trabalho com suas contribuições, lições aprendidas e trabalhos futuros.

Este trabalho constituiu da exploração de uma solução de processamento remoto para interação com HPM através de tablets. Foi desenvolvida uma aplicação de ambiente arquitetônico 3D que representou adequadamente uma HPM capaz de utilizar o potencial de *hardware* de última geração.

A construção do ambiente 3D motivou o uso e aperfeiçoamento de técnicas de modelagem 3D que possibilitaram com sucesso a geração de imagens em tempo real com qualidade semelhante à de imagens arquitetônicas pré-processadas.

A implementação e configuração da arquitetura de processamento remoto foi feita com o uso de ferramentas e técnicas disponíveis no mercado e utilizadas nas principais soluções atuais de *Cloud Gaming*, tornando a solução explorada compatível com esses serviços sem a necessidade de grandes adaptações.

Foram desenvolvidas duas opções de interface com a aplicação 3D. A primeira delas foi baseada nas formas de interações mais tradicionais em jogos e aplicações 3D para dispositivos móveis com tela *touchscreen.* A segunda foi baseada em técnicas estudadas e desenvolvidas com o intuito de se mitigar os efeitos da latência na experiência do usuário.

Foram realizados testes que indicaram boa capacidade da Solução Alternativa em mitigar os efeitos negativos da latência sobre o usuário. Obteve-se boa avaliação dos usuários em relação à qualidade visual das imagens da HPM desenvolvida comparada com as de imagens geradas por pré-processamento. E constatou-se que a solução é capaz de prover as funcionalidades esperadas por uma aplicação de visualização arquitetônica interativa segundo a opinião dos usuários.

Apesar da HPM neste trabalho ser específica para arquitetura, as técnicas e soluções utilizadas para viabilizá-la em uma arquitetura de processamento remoto com latência podem ser aproveitadas em aplicações semelhantes como: simulação

de museus, showrooms, laboratórios, salas de aula, locais históricos, etc. Por ser compatível com ferramentas já disponíveis no mercado, essa solução pode tirar proveito de serviços como o Amazon AppStream (AMAZON, 2015b) e colocada em prática sem a necessidade de grandes adaptações como as requeridas por soluções que propõe novos padrões ou alternativas ao *streaming* de vídeo e recursos não disponibilizados pelos principais *Game Engines*. A solução pode também ser utilizada em rede local, desde que os servidores possuam placas de vídeo compatíveis com o Nvidia GameStream.

Os testes com usuários revelaram-se de uma importância além da esperada durante o planejamento deste trabalho. As sessões de entrevistas forneceram opiniões valiosas que permitiram a visualização de problemas que poderiam ter sido evitados em etapas anteriores do desenvolvimento. A inclusão de protótipos para testes de usuário durante a concepção da interface poderia ter resultado em soluções melhores.

Há aspectos de uma solução de processamento remoto de aplicações 3D que não puderam ser abordados neste trabalho. A interface da Aplicação Cliente, por exemplo, foi desenvolvida especificamente para *tablets*. Apesar dos conceitos básicos de interações desenvolvidos não dependerem do tamanho da tela, há considerações importantes a serem feitas sobre a distribuição dos elementos de interface em áreas menores para tornar a interface responsiva que podem ser abordados em outros trabalhos.

Trabalhos que avaliaram o impacto da rede em jogos *multiplayer* e jogos de *Cloud Gaming* mostram que problemas de rede como *jitter* e perda de pacotes não afetam significativamente a experiência do usuário se comparados com alta latência. Porém, há problemas de rede móvel como a queda de conexões e restrições de banda com os quais uma solução de processamento remoto de HPM deve estar preparada para lidar. Esse aspecto pode ser abordado em trabalhos futuros.

Como notado no item 4.3.2, os menus da Solução Alternativa de interface podem conter diversos tipos de mídia processada localmente para complementar a experiência de interação com a HPM. Vídeos panorâmicos, serviços *web* ou gráficos 3D compatíveis com o *hardware* do Dispositivo Cliente são exemplos de conteúdo que pode ser explorado nesse tipo de combinação e requerem o estudo e desenvolvimento de formas de sincronização e implementação que podem ser explorados em outros trabalhos.

Por fim, o aspecto de escalabilidade da solução também abre espaço para estudos futuros, como o gerenciamento de diversos Dispositivos Clientes conectados com diversos Servidores e recursos de identificação de usuário e armazenamento de alterações feitas nos ambientes.

REFERÊNCIAS

AMAZON. **AWS Gaming**. Disponível em: <https://aws.amazon.com/pt/gaming/>. Acesso em: 10 fev. 2016a.

AMAZON. **AWS Amazon AppStream**. Disponível em: <https://aws.amazon.com/pt/appstream/>. Acesso em: 13 fev. 2016b.

ANDROID. **DrawerLayout - Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/reference/android/support/v4/widget/DrawerLayout.html>. Acesso em: 15 fev. 2016.

AUTODESK. **Software de modelagem e renderização 3ds Max 2016**. Disponível em: <http://www.autodesk.com.br/products/3ds-max/overview>. Acesso em: 22 fev. 2016a.

AUTODESK. **Adaptable File Format For 3D Animation Software**. Disponível em: <http://www.autodesk.com/products/fbx/overview>. Acesso em: 22 fev. 2016b.

BADASHIAN, A. S.; MAHDAVI, M.; POURSHIRMOHAMMADI, A.; NEJAD, M. M. Fundamental Usability Guidelines for User Interface Design. In: 2008 International Conference on Computational Sciences and Its Applications, **Anais**...IEEE, jun. 2008.

BEIGBEDER, T.; COUGHLAN, R.; LUSHER, C.; PLUNKETT, J.; AGU, E.; CLAYPOOL, M. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003®. In: Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04 Network and system support for games - SIGCOMM 2004 Workshops, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, ago. 2004.

BEKERMAN, R. **Epic Games Wants Architects to Make Unreal Engine More Lifelike**. Disponível em: <http://www.engadget.com/2015/06/04/epic-games-wants-architects-to-make-unreal-engine-more-lifelike/>. Acesso em: 13 fev. 2016.

CHEN, K.-T.; HUANG, P.; LEI, C.-L. How Sensitive Are Online Gamers to Network Quality? **Communications of the ACM**, v. 49, n. 11, p. 34, 1 nov. 2006.

CHOY, S.; WONG, B.; SIMON, G.; ROSENBERG, C. A hybrid edge-cloud architecture for reducing on-demand gaming latency. **Multimedia Systems**, v. 20, n. 5, p. 503–519, 11 abr. 2014.

CLAYPOOL, M.; FINKEL, D. The effects of latency on player performance in cloud-based games. In: 2014 13th Annual Workshop on Network and Systems Support for Games, **Anais**...IEEE, dez. 2014.

CRYTEK. **CryENGINE**. Disponível em: <http://www.crytek.com/cryengine>.

DICK, M.; WELLNITZ, O.; WOLF, L. Analysis of factors affecting players' performance and perception in multiplayer games. In: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games - NetGames '05, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 10 out. 2005.

DIEPSTRATEN, J.; GORKE, M.; ERTL, T. Remote line rendering for mobile devices. In: Proceedings Computer Graphics International, 2004., Crete. **Anais**... Crete: IEEE, 2004.

DYNAMICS, V. **Visual Dynamics VRay**. Disponível em: <https://www.vray.com/>. Acesso em: 22 fev. 2016.

EDUCAÇÃO, M. da. Todos Pela Educação - Governo impulsiona uso de tablet na escola. 2014.

EPIC GAMES. **Unreal Engine UDK**. Disponível em: <http://www.unrealengine.com/udk/>.

EPIC GAMES. **Unreal Engine 4**. Disponível em: <http://www.unrealengine.com/unreal_engine_4/>.

EPIC GAMES. **Winners of the Vineyard Challenge Revealed**. Disponível em: <https://www.unrealengine.com/blog/winners-of-the-vineyard-challenge-revealed>. Acesso em: 13 fev. 2016a.

EPIC GAMES. **Distance Field Ambient Occlusion**. Disponível em: <https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/DistanceFieldAmbientOcclusion/index.html>. Acesso em: 14 fev. 2016b.

EVERMOTION. **Evermotion Store**. Disponível em: <http://www.evermotion.org/shop/show_product/scene-02-aic01/12396>. Acesso em: 14 fev. 2016.

FORBES. **Nvidia Increases Desktop GPU Market Share Again During Q2, Despite Multiple AMD Radeon Releases - Forbes**. Disponível em: <http://www.forbes.com/sites/jasonevangelho/2015/08/19/nvidia-increases-desktop-gpu-market-share-again-despite-multiple-amd-radeon-releases/#57d4d0c5250e>. Acesso em: 13 fev. 2016a.

FORBES. **Apple iOS And Google Android Smartphone Market Share Flattening: IDC - Forbes**. Disponível em: <http://www.forbes.com/sites/dougolenick/2015/05/27/apple-ios-and-google-android-smartphone-market-share-flattening-idc/2/#37b5758194f1>. Acesso em: 15 fev. 2016b.

GAIKAI. **Gaikai**. Disponível em: <https://www.gaikai.com/>. Acesso em: 20 fev. 2016.

GARBER, L. GPUs Go Mobile. **IEEE Computer**, v. 46, n. 2, p. 16–19, 1 fev. 2013.

GHILETIUC, J.; FÄRBER, M.; BRÜDERLIN, B. Real-time remote rendering of

large 3D models on smartphones using multi-layered impostors. In: Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications - MIRAGE '13, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 6 jun. 2013.

GODOY, A. P. **Diretrizes para promover o uso de dispositivos com limitações computacionais na interação com mídias sintetizadas remotamente**. 2014. 40f. Dissertação (Mestrado) - Universidade Federal de São Carlos, São Carlos, 2014, 2014.

GOMES SOARES, L.; MORENO, M.; SALLES SOARES NETO, C.; MORENO, M. Ginga-NCL: Declarative middleware for multimedia IPTV services. **IEEE Communications Magazine**, v. 48, n. 6, p. 74–81, jun. 2010.

GOOGLE. **Google Play**. Disponível em: <https://play.google.com/store/apps/dev?id=5700313618786177705>. Acesso em: 15 fev. 2016.

HUANG, J.; QIAN, F.; GERBER, A.; MAO, Z. M.; SEN, S.; SPATSCHECK, O. A close examination of performance and power characteristics of 4G LTE networks. In: Proceedings of the 10th international conference on Mobile systems, applications, and services - MobiSys '12, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 25 jun. 2012.

IDC. **IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012**. Disponível em: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em: 15 fev. 2016.

ITU-T. **P.800.2 : mean opinion score interpretation and reporting**. [s.l: s.n.]

JAGT. **Clumsy**. Disponível em: <https://jagt.github.io/clumsy/>. Acesso em: 21 fev. 2016.

JARSCHEL, M.; SCHLOSSER, D.; SCHEURING, S.; HOSSFEL, T. An Evaluation of QoE in Cloud Gaming Based on Subjective Tests. In: 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Seoul. **Anais**... Seoul: IEEE, jun. 2011.

LAPEDES, D. N. (ed. . **Mcgraw-hill dictionary of scientific and technical terms**. 6. ed. [s.l.] McGraw-Hill Book Co.,New York, NY, 2002.

LEE, K.; CHU, D.; CUERVO, E.; KOPF, J.; WOLMAN, A.; DEGTYAREV, Y.; GRIZAN, S.; FLINN, J. Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. **GetMobile: Mobile Computing and Communications**, v. 19, n. 3, p. 14–17, 23 dez. 2015.

MALONE, T. W. Heuristics for designing enjoyable user interfaces. In: Proceedings of the 1982 conference on Human factors in computing systems - CHI '82, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 15 mar. 1982.

MARTIN, C. **What is PlayStation Now?** Disponível em:

<http://www.pcadvisor.co.uk/feature/game/playstation-now-release-date-price-supported-devices-features-uk-launched-october-2015-3497234/>. Acesso em: 10 fev. 2016.

MOLICH, R.; NIELSEN, J. Improving a human-computer dialogue. **Communications of the ACM**, v. 33, n. 3, p. 338–348, 1 mar. 1990.

MOONLIGHT. **Moonlight Game Streaming**. Disponível em: <http://moonlight-stream.com/>. Acesso em: 15 fev. 2016.

NIELSEN, J. **Usability engineering**. 1ª. ed. San Francisco: Morgan Kaufmann Publishers Inc., 1994a.

NIELSEN, J. Enhancing the explanatory power of usability heuristics. In: Proceedings of the SIGCHI conference on Human factors in computing systems celebrating interdependence - CHI '94, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 24 abr. 1994b.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces. In: Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 1 mar. 1990.

NVIDIA. **Nvidia GeForce GRID**. Disponível em: <http://www.geforce.com/whats-new/articles/geforce-grid>. Acesso em: 22 fev. 2016.

NVIDIA. **Cloud Gaming – Gaming as a Service (GaaS)**. Disponível em: <http://www.nvidia.com/object/cloud-gaming.html>. Acesso em: 10 fev. 2016a.

NVIDIA. **Nvidia Game Stream**. Disponível em: <https://shield.nvidia.com/game-stream>. Acesso em: 13 fev. 2016b.

NVIDIA. **Nvidia SHIELD**. Disponível em: <https://shield.nvidia.com/>. Acesso em: 15 fev. 2016c.

NVIDIA. **Enabling New Algorithms and Superior Image Quality**. Disponível em: <http://www.geforce.com/hardware/technology/vxgi/technology>. Acesso em: 14 fev. 2016d.

NVIDIA. **Nvidia GameWorks**. Disponível em: <https://developer.nvidia.com/gameworks>. Acesso em: 14 fev. 2016e.

OTOY. **OTOY X.IO**. Disponível em: <https://home.otoy.com/stream/xio/>. Acesso em: 13 fev. 2016.

PANTEL, L.; WOLF, L. C. On the impact of delay on real-time multiplayer games. In: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '02, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 12 maio 2002.

PATRICK, A. S.; SINGER, J.; CORRIE, B.; NOEL, S.; EL KHATIB, K.; EMOND, B.; ZIMMERMAN, T.; MARSH, S. A QoE sensitive architecture for advanced

collaborative environments. In: First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks, **Anais**...IEEE Comput. Soc, 2004.

PLANTE, C. **Why video game engines may power the future of film and architecture - The Verge**. Disponível em: <http://www.theverge.com/2015/3/4/8150057/unreal-engine-4-epic-games-tim-sweeney-gdc-2015>. Acesso em: 10 fev. 2016.

QUAX, P.; MONSIEURS, P.; LAMOTTE, W.; DE VLEESCHAUWER, D.; DEGRANDE, N. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In: Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04 Network and system support for games - SIGCOMM 2004 Workshops, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 30 ago. 2004.

RABBITMQ. **RabbitMQ**. Disponível em: <https://www.rabbitmq.com/>. Acesso em: 14 fev. 2016.

RABELLO, G. **Transformação de ambiente arquitetônico virtual de alto realismo em objeto hpm**. 2013. Universidade Federal de São Carlos, 2013.

SAVAGE, P. **Unreal Engine 4 architectural visualisation videos are staggeringly realistic - PC Gamer**. Disponível em: <http://www.pcgamer.com/unreal-engine-4-architectural-visualisation-videos-are-staggeringly-realistic/>. Acesso em: 13 fev. 2016.

SHEA, R.; NGAI, E. C.-H. Cloud Gaming: Architecture and Performance. **IEEE Network**, v. 27, n. 4, p. 16–21, 2013.

SHNEIDERMAN, B. **Designing the user interface: strategies for effective human-computer interaction**. 1ª. ed. Boston: Addison-Wesley Longman Publishing Co., Inc., 1986.

SILLER, M.; WOODS, J. Improving Quality of Experience for Multimedia Services by QoS arbitration on QoE Framework. In: Procedures of the 13th Packed Video Workshop 2003, **Anais**...2003.

SOMMERS, J.; BARFORD, P. Cell vs. WiFi: on the performance of metro area mobile connections. In: Proceedings of the 2012 ACM conference on Internet measurement conference - IMC '12, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 14 nov. 2012.

STREIJL, R. C.; WINKLER, S.; HANDS, D. S. Mean opinion score (MOS) revisited: methods and applications, limitations and alternatives. **Multimedia Systems**, v. 22, n. 2, p. 213–227, 28 dez. 2014.

STUART, K. **Photorealism - the future of video game visuals**. Disponível em: <http://www.theguardian.com/technology/2015/feb/12/future-of-video-gaming-visuals-nvidia-rendering>. Acesso em: 10 fev. 2016.

TELER, E.; LISCHINSKI, D. Streaming of Complex 3D Scenes for Remote Walkthroughs. **Computer Graphics Forum**, v. 20, n. 3, p. 17–25, set. 2001.

UNITY. **Unity Game Engine**. Disponível em: <https://unity3d.com/pt>. Acesso em: 13 fev. 2016.

VIEL, C. C.; MELO, E. L.; GODOY, A. P.; DIAS, D. R. C.; TREVELIN, L. C.; TEIXEIRA, C. A. C. Multimedia presentation integrating interactive media produced in real time with high performance processing. In: Proceedings of the 18th Brazilian symposium on Multimedia and the web - WebMedia '12, New York, New York, USA. **Anais**... New York, New York, USA: ACM Press, 15 out. 2012.

VIEL, C.; MELO, E.; TEIXEIRA, C. A. C.; TREVELIN, L. C. RV-MTV: Framework para Interação Multimodal com Aplicações de Realidade Virtual em TV Digital e Dispositivos Móveis. In: Proceedings of the 18th Brazilian symposium on Multimedia and the web - WebMedia 11, **Anais**...2011.

WINKLER, S. On the properties of subjective ratings in video quality experiments. In: 2009 International Workshop on Quality of Multimedia Experience, **Anais**...IEEE, jul. 2009.